VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Ohodnocení grafů

Graph labelings

2018 Tomáš Michna

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Tomáš Michna**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Ohodnocení grafů
Graph labelings

Jazyk vypracování: čeština

Zásady pro vypracování:

Ohodnocení grafů patří mezi klasická témata teorie grafů, existuje bezpočet aplikací. Důležitou kapitolu tvoří problémy existence ohodnocení s předepsanými parametry: předepsaná množina hodnot, množina vah, vztah vah a dalších parametrů grafu.
Práce bude rozdělena do několika částí:
- výběr jednoho nebo několika zvolených typů ohodnocení
- studium a zpracování přehledu známých výsledků
- rozbor vlastností ohodnocení, známých metod konstrukce
- návrh vlastní konstrukce nebo algoritmu
- případně implementace

Tato práce může být zaměřena teoreticky s důrazem na rozbor a návrh metod ohodnocení, ale i prakticky s důrazem na kvalitní implementaci či na paralelní implementaci.
Výsledky práce budou využity v navazujících pracech v oblasti grafových ohodnocení.

Seznam doporučené odborné literatury:

- W. Wallis: Magic graphs, Birkhauser 2000.
- M. Bača, M. Miller: Super Edge-Antimagic Graphs, Brown, Walkers Press, 2008.
- J.A. Gallian, A dynamic survey of graph labeling, The Electronic Journal of Combinatorics, DS 6 (2011).
- odborné články a texty podle pokynů vedoucího

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Petr Kovář, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

.....

Rád bych na tomto místě poděkoval svému vedoucímu bakalářské práce,
panu Doc. Mgr. Petru Kovářovi, Ph.D. za cenné rady, připomínky a za čas, který mi věnoval po celou dobu vedení této bakalářské práce, hlavně pak za nadšení, které mi předal.

Abstrakt

V této bakalářské práci se zabýváme oblastí zvané ohodnocení grafů, ve které zkoumáme 3-pravidelné a 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením. Zmiňujeme současné poznatky v tykající se ohodnocení r-pravidelných grafů. Přicházíme s novými výsledky, které jsou shrnuty do matematických vět o existenci 3-pravidelných grafů, kde počet vrcholů je násobkem 8 a existenci 4-pravidelných grafů, kde počet vrcholů je násobkem 12. Uvádíme zde i vlastní polynomiální algoritmus pro hledaní všech r-pravidelných grafů.

Klíčová slova: 3-pravidelné grafy, 4-pravidelné grafy, rozšířené hendikepové ohodnocení, algoritmus hledání r-pravidelných grafů s rozšířeným hendikepovým ohodnocením

Abstract

In this bachelor thesis we deal with the area called graph labeling. We examine 3-regular graphs and 4-regular graphs with extended handicap labeling. We mention the current knowledge about r-regular graphs with extended handicap labeling. We provide new results, which are summarized into mathematical theorems about existence of 3-regular graphs, where the number of vertices must be a multiple of 8 and about existence of 4-regular graphs, where number of vertices must be a multiple of 12. We also describe a new polynomial algorithm for finding all r-regular graphs with extended handicap labeling.

Key Words: 3-regular graphs, 4-regular graphs, extended handicap labeling, algorithm for search r-regular graphs with extended handicap labeling

Obsah

Seznam použitých zkratek a symbolů

$G = (V, E)$	Graf G s množinou vrcholů V a množinou hran E
$V(G)$	Množina vrcholů grafu G
$E(G)$	Množina hran grafu G
$N_G(v)$	Množina sousedních vrcholů vrcholu v v grafu G
$w(v)$	Váha vrcholu v v rozšířeném hendikepovém ohodnocení
C_n	Cyklus na n vrcholech
K_n	Kompletní graf na n vrcholech
Q_n	Hyperkrychle řádu n
$H_{n,n}$	Crown graf řádu n
r	Pravidelnost grafu
ℓ	Konstanta v rozšířeném hendikepovém ohodnocení
y	Chybějící číslo v posloupnosti labelů rozšířeného hendikepového ohodnocení
CPU	Centrální procesorová jednotka
x_{CPU}	Počet centrálních procesorových jednotek

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Ohodnocení grafů je jednou z disciplín teorie grafů. Toto odvětví zkoumá zákonitosti při injekci množin vrcholů grafu do intervalu přirozených čísel, nebo injekci množin hran do intervalu přirozených čísel či kombinací těchto zobrazení. První zmínka o grafovém ohodnocení je připisována českému matematikovi Jiřímu Sedláčkovi, který v roce 1962 publikoval článek s problémem 27 v "Theory of graphs and its applications". Čímž se toto odvětví teorie grafů dostalo více do podvědomí matematikům. Jedná se sice o relativně mladou disciplínu, ale její výsledky nacházejí uplatnění i v dnešním světě. Vhodným příkladem může být rozvrh plánované herní soutěže např. turnaj fotbalových týmů. Cílem je naplánovat herní duely týmů tak, aby vyhovovaly různým kriteriím např. neúplný turnaj týmů. Toto kritérium můžeme chápat tak, že silný tým se utká se stejně silným kandidáty a slabý tým se utká se stejně silnými týmy. Jedním z pohledů, jak vyřešit tento problém nabízí ohodnocení grafů. Mým úkolem bylo prozkoumat oblast 3-pravidelných a 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením a sestrojit program pro hledání 4-pravidelných grafů s tímto ohodnocením.

2 Základní pojmy a definice

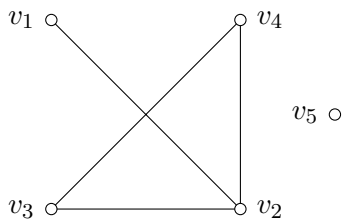
2.1 Jednoduchý graf

Abychom mohli zkoumat soutěžní turnaje a vytvářet rozvrhy zápasů, zavedeme si nejprve matematickou strukturu grafu. Nejedná se o graf funkce, ale o diskrétní matematickou strukturu pomocí které, budeme modelovat zápasy mezi soutěžními družstvy. Výhodou této struktury je snadná vizuální interpretace v diagramu, kterou si vzápětí ukážeme. Pojem grafu zavedl Leonhard Euler roku 1736, když řešil úlohu sedmi mostů ve městě Královci v dnešním Kaliningradu. Od té doby se začalo formovat odvětví matematiky zvané teorie grafů a každým rokem přibývá nespočet nových aplikací, využívajících poznatků z této oblasti k řešení i těch nesložitějších problémů, ať už se jedná o hledání nejkratší trasy pomocí GPS, hledání webové stránky přes vyhledávač, či modelování složitých vztahů mezi proteiny v bioinformatice. My však poznatky této vědy využijeme ke zkoumání neúplných soutěžních turnajů, kterými se budeme zabývat v této bakalářské práci.

Definice 1 *Jednoduchý neorientovaný (obyčejný) graf G je uspořádaná dvojice (V, E), kde V je neprázdná množina vrcholů a E je množina dvouprvkových podmnožin V. Prvkům v E říkáme hrany.*

Poznámka 1 V této práci budeme nadále pracovat pouze s tímto typem grafů. Graf budeme obvykle značit velkými tiskacími písmeny G nebo H. Malými písmeny v, v_i, u budeme značit vrcholy daného grafu.

Množině $V(G)$ budeme říkat množina vrcholů grafu G a množině $E(G)$ budeme říkat hranová množina grafu G. Uveďme nyní příklad grafu G, který je znázorněn na Obrázku 1. Vrcholová množina grafu G je rovna $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$. Hranová množina grafu G je $E(G) = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\}$. Zaveďme si úmluvu, že hranu $e = \{u, v\}$ budeme v textu nadále značit zkráceným zápisem $e = uv$. Řádem grafu G budeme rozumět velikost množiny $V(G)$ kterou budeme značit $|V(G)| = n$. Obdobným způsobem budeme značit počet hran grafu G takto $|E(G)|$.



Obrázek 1: Jednoduchý graf G

Pojmem *jednoduchý graf* chápeme v tomto textu takový graf G, ve kterém se nevykytuje žádná hrana $e = v_i v_i$, která se nazývá smyčka. Taktéž se v jednoduchém grafu nemůže vyskytnout hrana $v_i v_j$ více krát, což by znamenalo, že graf G obsahuje multihrany.

Definice 2 *Mějme jednoduchý graf G, potom dva vrcholy $u, v \in V(G)$ nazvěme závislé (sousední), pokud v hranové množině $E(G)$ existuje hrana uv, v opačném případě nazvěme vrcholy u a v nezávislé. Množinu všech sousedních vrcholů k vrcholu u v grafu G budeme značit $N_G(u)$.*

Definice 3 *Mějme dán jednoduchý graf $G = (V, E)$, tento graf nazveme triviálním, jestliže řád grafu G je $|V(G)| = 1$.*

Využijme graf G z Obrázku 1, kde množina sousedních vrcholů k vrcholu v_2 je $N_G(v_2) = \{v_1, v_3, v_4\}$ a množina sousedních vrcholů k vrcholu v_5 je $N_G(v_5) = \emptyset$. Dále zaveďme definici stupně vrcholu.

Definice 4 *[2] Stupeň vrcholu u je počet hran, se kterými je vrchol u incidentní a značí se* $\deg(u)$.

Využijme opět příkladu grafu G z Obrázku 1, kde stupeň vrcholu v_2 je $\deg(v_2) = 3$ a stupeň vrcholu v_5 je $\deg(v_5) = 0$. Dále uveďme pojem *stupňová posloupnost grafu G*, kterou využijeme ve Větě Havla-Hakimiho.

Definice 5 *[2] Mějme dán jednoduchý graf G s vrcholy $v_1, v_2, \ldots, v_n$. Posloupnost* $(\deg(v_1), \deg(v_2), \ldots, \deg(v_n))$ *nazýváme stupňovou posloupností grafu G, případně skóre grafu G. Nerostoucí posloupnost $D = (d_1, d_2, \ldots, d_n)$ se nazývá grafová, je-li stupňovou posloupností nějakého grafu.*

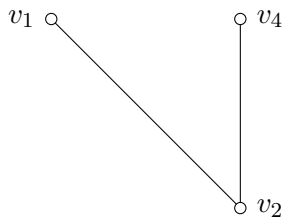
Věta 1 *[2]* ***Havel-Hakimi***
Nechť $D = (d_1, d_2, \ldots, d_n)$ je nerostoucí posloupnost a nechť D' vznikne přeuspořádáním posloupnosti $(d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ na nerostoucí posloupnost. Potom platí, že D je grafová posloupnost netriviálního grafu právě tehdy, když D' je grafová posloupnost.

Důkaz Věty 1 je uveden v učebním textu Teorie grafů [2]. Dále uvedeme pojem *podgraf grafu G* a s ním související pojem *faktor grafu G*. Tyto pojmy pak využijeme v konstruktivních důkazech 3-pravidelných a 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením.

Definice 6 *[2] Mějme dán graf $G = (V, E)$. Řekneme, že graf $H = (V', E')$ je podgrafem grafu G, jestliže $V' \subseteq V$ a současně $E' \subseteq E$.*

Příkladem podgrafu ke grafu G z Obrázku 1 může být graf $H = (V', E')$, kde $V' = \{v_1, v_2, v_4\}$ a $E' = \{v_1 v_2, v_2 v_4\}$. Graf H je zobrazen na Obrázku 2.

Definice 7 *Mějme jednoduchý graf $G = (V, E)$, potom jeho podgraf $H = (V', E')$, nazveme faktorem grafu G, jestliže $V' = V$ a současně $E' \subseteq E$.*



Obrázek 2: Podgraf H grafu G

Dalším z pojmů, které jsou nezbytné k pochopení důkazů uvedených v sekci 4.3 a 4.4, je pojem *komponenty souvislosti*. Dříve než tento pojem uvedeme, nadefinujme pojmy *sledu* a *souvislosti*.

Definice 8 *[2] Sled v grafu G je taková posloupnost vrcholů a hran $(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$, že hrana e_i má koncové vrcholy v_{i-1} a v_i pro všechna $i = 1, 2, \ldots, n$. Sled $(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$ se nazývá (v_0, v_n)-sled.*

Definice 9 *[2] Řekneme, že vrchol v je dosažitelný z vrcholu u v grafu G, jestliže v G existuje (u, v)-sled. Graf G je souvislý, jestliže pro každou dvojici vrcholů u, $v \in V(G)$ existuje (u, v)-sled. V opačném případě je graf nesouvislý.*

Definice 10 *[2] Řekneme, že L je komponenta grafu G, jestliže je L souvislý podgraf grafu G a současně pro každý souvislý podgraf W grafu G platí, že W je podgraf L a nebo vrcholové množiny W a L jsou disjunktní. Počet komponent grafu G značíme $\omega(G)$.*

Pro názornost mějme náš graf G z Obrázku 1, kde $L = (V_L, E_L)$ a $W = (V_W, E_W)$ jsou souvislé podgrafy grafu G, kde $V_L = \{v_1, v_2, v_3, v_4\}$, $E_l = \{v_1v_2, v_2v_3, v_2v_4, v_3v_4\}$ a $V_W = \{v_5\}$, $E_W = \emptyset$. Potom grafu G má počet komponent roven $\omega(G) = 2$.

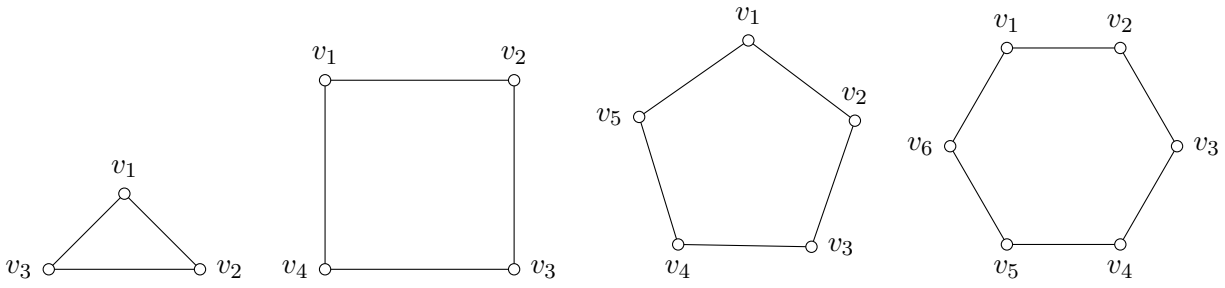
Dále zde uveďme známe typy grafů zvané *kompletní grafy* a *cykly*.

Definice 11 *Mějme jednoduchý graf $G = (V, E)$. Nazvěme tento graf cyklem, jestliže $V = \{v_1, v_2, \ldots, v_n\}$, $|V(G)| \geq 3$ a současně platí, že $E(G) = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\}$. Potom tento graf budeme značit C_n, kde $n = |V(G)|$.*

Pro názornost uvádíme cykly C_3, C_4, C_5 a C_6 na Obrázku 3.

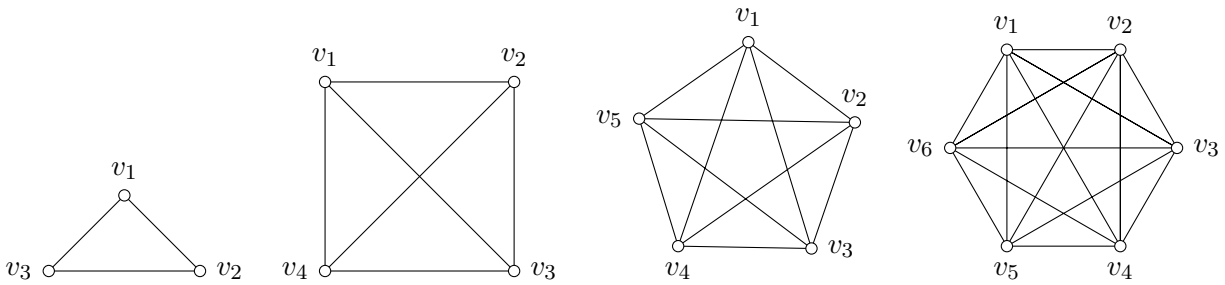
Definice 12 *[1] Graf s označením K_n nazveme kompletním grafem, pokud v tomto grafu jsou každé dva vrcholy sousední.*

Pro názornost uvádíme kompletní grafy K_3, K_4, K_5 a K_6 na Obrázku 4.



(a) Cyklus C_3 (b) Cyklus C_4 (c) Cyklus C_5 (d) Cyklus C_6

Obrázek 3: Cykly



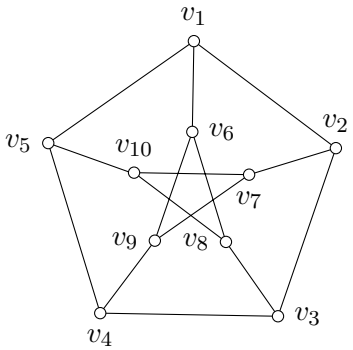
(a) Kompletní graf K_3 (b) Kompletní graf K_4 (c) Kompletní graf K_5 (d) Kompletní graf K_6

Obrázek 4: Kompletní grafy

Dalším pojmem, který zmíníme, bude *r-pravidelný graf.*

Definice 13 *[1] Pokud v grafu G $\forall v_i \in V(G)$ platí* $\deg(v_i) = r$*, pak nazveme tento graf G r-pravidelným (r-regulárním).*

Tento pojem je zvlášť pro nás důležitý, protože budeme hledat 3-pravidelné a 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením. Pro příklad uvádíme 3-pravidelný graf na deseti vrcholech, který se nazývá Petersenův graf na Obrázku 5.



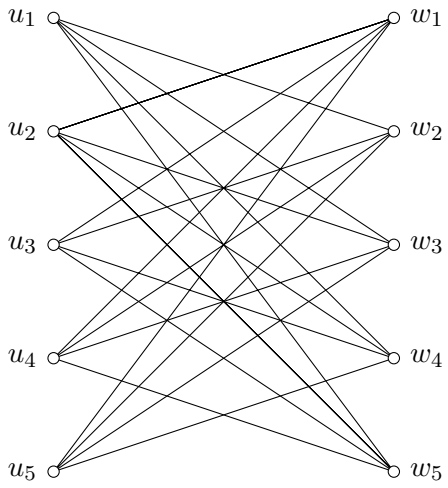
Obrázek 5: Petersenův graf

2.2 Crown graf

Uvedeme zde definici n-crown grafu. Důvod proč tento graf uvádíme, je jeho výskyt v nalezených 4-pravidelných grafech s rozšířeným hendikepovým ohodnocením na 22 a 30 vrcholech, který nalezl počítačový program implementující Algoritmus 8 a detailně je popsán v sekci 5.

Definice 14 *[7] Mějme jednoduchý neorientovaný graf $G = (V, E)$ s $2n$ vrcholy kde $n \geq 3$, jehož vrcholová množina je sjednocením dvou disjunktních neprázdných partit U a W, $V(G) = U \cup W$. Každá z partit U a W má právě n vrcholů, tj. $|U| = |W| = n$. Hranová množina grafu G je $E(G) = \left\{ \bigcup_{i=1}^{n} \bigcup_{j=1}^{n} u_i w_j,\ u_i \in U \wedge w_j \in W \ \wedge \ i \neq j \right\}$. Potom takový graf nazveme n-crown graf a budeme ho značit $H_{n,n}$.*

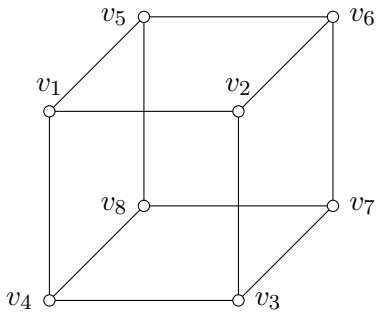
Z definice vidíme, že se jedná o bipartitní graf. Pro názornost uvedeme 5-crown graf na 10 vrcholech na Obrázku 6.



Obrázek 6: Pořadí označených vrcholů v 5-crown grafu

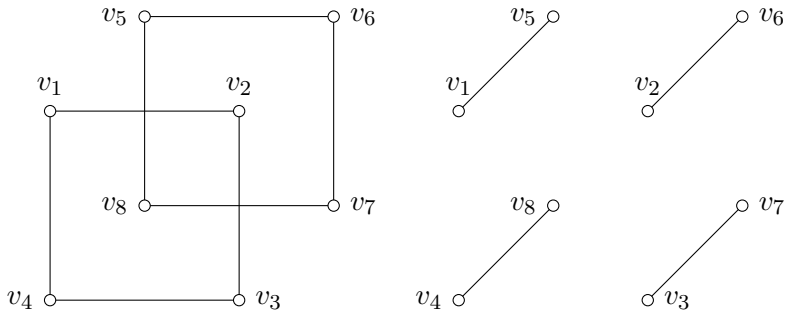
2.3 Hyperkrychle Q_3

Popíšeme základní faktorizaci hyperkrychle Q_3. Jedná se o 3-pravidelný nebo též kubický graf na osmi vrcholech, který je zobrazen na Obrázku 7.



Obrázek 7: Pořadí označených vrcholů v grafu hyperkrychle Q_3

Jednou z klíčových vlastností grafu hyperkrychle $Q_3 = (V, E)$, kterou využijeme v důkazu Věty 11, je jeho hranově disjunktní rozklad na dva faktory Q'_3 a Q''_3. Faktor $Q'_3 = (V, E_1)$ tvoří dvě disjunktní komponenty C_4. Faktor $Q''_3 = (V, E_2)$ je tvořen čtyřmi disjunktními komponentami K_2. Můžeme si všimnout, že $E_2 = E \setminus E_1$. Názorné zobrazení faktorů Q'_3 a Q''_3 je v Obrázku 8.

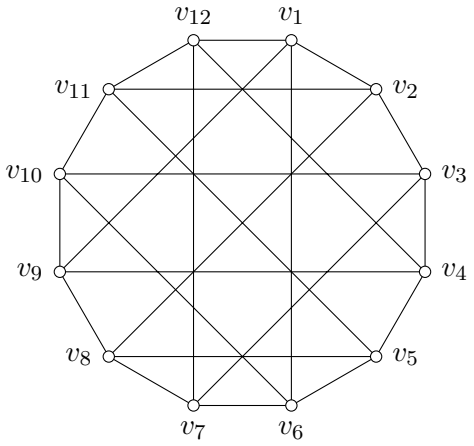


(a) Hranově disjunktní faktor Q'_3 (b) Hranově disjunktní faktor Q''_3

Obrázek 8: Hranově disjunktní rozklad grafu Q_3 na faktory Q'_3 a Q''_3

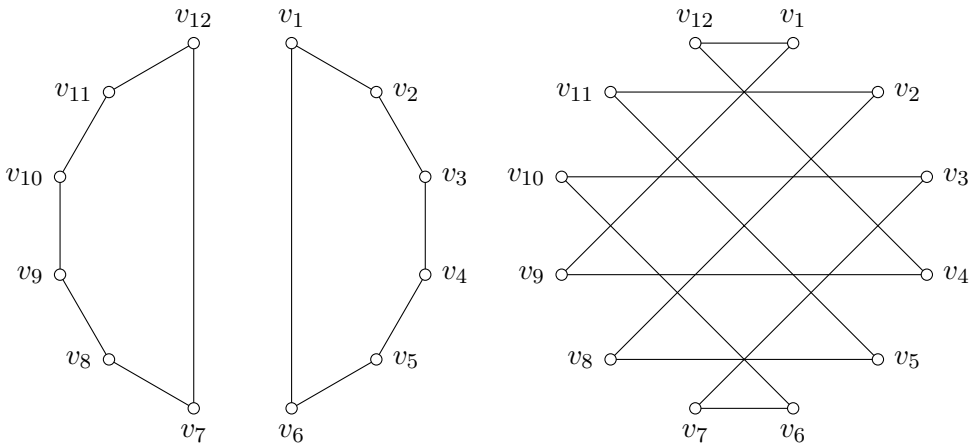
2.4 Nalezený 4-pravidelný graf H na 12 vrcholech

Nyní popíšeme základní graf H. Jednou z klíčových vlastností grafu $H = (V, E)$ je jeho hranově



Obrázek 9: Pořadí označených vrcholů v grafu H

disjunktní rozklad na dva faktory H' a H''. Faktor $H' = (V, E_1)$ tvoří dvě disjunktní komponenty C_6. Faktor $H'' = (V, E_2)$ je tvořen třemi disjunktními komponentami C_4. Můžeme si všimnout že $E_2 = E \setminus E_1$. Názorné zobrazení faktorů H' a H'' se nachází v Obrázku 10.



(a) Hranově disjunktní faktor H' grafu H (b) Hranově disjunktní faktor H'' grafu H

Obrázek 10: Hranově disjunktní rozklad grafu H na faktory

3 Hendikepové ohodnocení

Poznámka 2 Pojem hendikepové ohodnocení překládáme z anglického termínu handicap labeling. Dále v textu budeme toto přídavné jméno hendikepové skloňovat podle vzoru mladý.

Definice 15 *[6] Mějme jednoduchý neorientovaný graf $G = (V, E)$, potom zobrazení*

$$f : V \to \{1, 2, \ldots, n\}$$

nazveme hendikepové ohodnocení, pokud existuje celé číslo ℓ, takové že splňuje podmínku pro

$$\forall v \in V : \sum_{u \in N_G(v)} f(u) = \ell + f(v).$$

Výraz $\sum_{u \in N(v)} f(u)$nazveme váhou $w_f(v)$ vrcholu v při hendikepovém ohodnocení. Prvkům množiny $\{1, 2, \ldots, n\}$ budeme říkat labely hendikepového ohodnocení.

Poznámka 3 Pojmem label budeme rozumět číslo z množiny $\{1, 2, \ldots, n\}$, které je v ohodnocení f přiřazeno k určitému vrcholu grafu G. Českým ekvivalentním výrazem pro anglické slovo label je slovní spojení ohodnocení vrcholu. Tento pojem se dá chápat dvěma způsoby. První způsob je ten, který jsme uvedli jak budeme slovu label rozumět. Ve vědecké komunitě zabývající se ohodnocením grafu, je tento prvek z množiny nazýván label, a proto jsme se rozhodli využít tohoto slova i v našem textu. Druhý způsob, jak můžeme chápat slovní spojení ohodnocení vrcholu je proces, při kterém určitému vrcholu v grafu G přiřadíme konkrétní číslo z množiny $\{1, 2, \ldots, n\}$. Ve vědecké komunitě, zabývající se ohodnocením grafu se vžilo pojmenování pro tento proces labeling. Nadále budeme skloňovat slovo label podle vzoru hrad.

Uveďme větu o váze v r-pravidelných grafech s hendikepovým ohodnocením.

Věta 2 *[3] V r-pravidelném hendikepovém grafu G na n vrcholech je váha každého vrcholu dána vztahem*

$$w(v_i) = \frac{(r-1)(n+1)}{2} + i.$$

Ze vztahu vidíme, že konstanta ℓ hendikepového ohodnocení pro r-pravidelné grafy na n vrcholech je

$$\ell = \frac{(r-1)(n+1)}{2}.$$

Uveďme větu o existenci r-pravidelných grafů s hendikepovým ohodnocením v závislosti na parametrech r a n.

Věta 3 *[3] Neexistuje žádný r-pravidelný hendikepový graf s n vrcholy, pokud r a n jsou sudá.*

Snadno nahlédneme, že ℓ by nebylo celé číslo, což není možné. Důkazy Vět 2 a 3 jsou také uvedené v diplomové práci [4] .

4 Rozšířené hendikepové ohodnocení

Této oblasti se věnoval Matěj Krbeček ve své diplomové práci [1]. Zavedl pojem rozšířeného hendikepového ohodnocení a podařilo se mu dokázat mnoho vět o rozšířeném hendikepovém ohodnocení na třídě 2-pravidelných grafů. Věty zde uvedeme. Důkazy těchto vět jsou uvedeny v diplomové práci Matěje Krbečka [1].

Poznámka 4 Následující pojem překládáme z anglického termínu extended handicap labeling, jako rozšířené hendikepové ohodnocení. V diplomové práci Matěje Krbečka [1] je tento pojem pojmenován jako upravené hendikepové ohodnocení. Jedná se o stejný termín jako rozšířené hendikepové ohodnocení.

Definice 16 *Mějme jednoduchý neorientovaný graf $G = (V, E)$ se sudým počtem vrcholů, potom zobrazení*

$$f : V \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n + 1\right\}$$

nazveme rozšířené hendikepové ohodnocení, pokud existuje celé kladné číslo ℓ, takové že splňuje podmínku

$$\forall v \in V : \sum_{u \in N_G(v)} f(u) = \ell + f(v).$$

Číslo $\sum_{u \in N(v)} f(u)$ budeme nazývat váhou vrcholu v při rozšířeném hendikepovém ohodnocení a značit tuto váhu budeme symbolem $w_f(v)$. Prvkům množiny $\{1, 2, \ldots, \frac{n}{2} - 1\frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n + 1\}$ budeme říkat labely rozšířeného hendikepového ohodnocení.

Věta 4 *[1] Mějme jednoduchý neorientovaný graf $G = (V, E)$, který má upravené hendikepové ohodnocení. Předpokládejme že:*

1. $x = y$,
2. $\frac{(r-1)(n+2)}{2n} + \frac{(x-ry)}{n} = 0$,

kde x je vynechaná váha, y vynechaný label, r pravidelnost grafu a n počet vrcholů grafu G. Potom konstanta ℓ je určena jednoznačně podle vztahu $\ell = \frac{(r-1)(n+2)}{2}$.

Díky této Větě 4 víme, jak vypočíst konstantu ℓ na r-pravidelných grafech s rozšířeným hendikepovým ohodnocením, tak aby labely tvořily aritmetickou posloupnost bez jednoho labelu, který je roven číslu $y = \frac{(n+2)}{2}$ a je zmíněn ve Větě 5.

Věta 5 *[1] Ze získaného vztahu pro konstantu ℓ vyplývá, že rozšířené hendikepové ohodnocení existuje pouze na grafech se sudým počtem vrcholů. Pokud by n bylo liché, muselo by být r také liché, aby konstanta ℓ vyšla celé číslo. Z principu sudosti víme, že neexistují grafy, které mají r i n současně liché. Dále jsme určili label vrcholu který, musíme vynechat. Jedná se o label vrcholu $\frac{n+2}{2}$ a z požadavku $x = y$ plyne, že vynechaná váha je právě vahou tohoto vrcholu.*

Dále uveďme přehlednou Tabulku 1, ve které shrneme možné existence r-pravidelných grafů s rozšířeným hendikepovým ohodnocením (RHO) a hendikepovým ohodnocením (HO) na n vrcholech. Symbolem x označme neexistneci příslušné kombinace, která vyplývá z neexistence takového grafu.

	r sudé	r liché
n sudé	RHO	RHO, HO
n liché	HO	x

Tabulka 1: Existence r-pravidelných grafů s daným ohodnocením

Důvodem vzniku rozšířeného hendikepového ohodnocení byl i fakt, že může existovat ohodnocení, ve kterém r i n je sudé což pro hendikepovému ohodnocení není možné dle Věty 3.

4.1 1-pravidelné grafy s rozšířeným hendikepovým ohodnocením

Matěj Krbeček ve své diplomové práci [1] zmiňuje, že 1-pravidelné grafy s rozšířeným hendikepovým ohodnocením neexistují. Uvedeme důkaz tohoto tvrzení.

Věta 6 *Neexistuje 1-pravidelný graf s rozšířeným hendikepovým ohodnocením na n vrcholech.*

Důkaz Díky Větě 5 má smysl se zabývat pouze grafy se sudým počtem vrcholů. Tuto větu dokážeme sporem. Předpokládejme, že existuje jednoduchý obyčejný neorientovaný 1-pravidelný graf G s rozšířeným hendikepovým ohodnocením na sudém počtu vrcholů n. Pak konstantu ℓ vypočteme ze vztahu, který je uveden ve Větě 4 a má hodnotu

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{0 \cdot (n+2)}{2} = 0.$$

Dále z Definice 16 rozšířeného hendikepového ohodnocení víme, že váha w pro každý vrchol grafu G je dána vztahem

$$\forall v \in V(G) : \sum_{u \in N_G(v)} f(u) = \ell + f(v) = 0 + f(v) = f(v).$$

Dostáváme pak rovnici pro každý vrchol grafu G

$$\forall v \in V(G) \wedge u \in N(v) : f(u) = \ell + f(v) = 0 + f(v) = f(v).$$

Aby nastala požadovaná rovnost ve výše uvedené rovnici, musí se vrchol u rovnat vrcholů v. To ovšem znamená, že každý vrchol grafu G má hranu vedoucí do sebe sama. Což je v rozporu s Definicí 16, že graf G je jednoduchý. □

4.2 2-pravidelné grafy s rozšířeným hendikepovým ohodnocením

Jak již bylo zmíněno v úvodu této kapitoly, oblastí zkoumání 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením se zabýval Matěj Krbeček ve své diplomové práci [1]. Uvedeme zde hlavní tvrzení, které se Matějovi Krbečkovi podařilo dokázat.

Věta 7 *[1] Mějme 2-pravidelný graf G s rozšířeným hendikepovým ohodnocením. Pak jsou splněny následující podmínky:*

1. *každá komponenta G je isomorfní s C_6,*
2. *počet vrcholů n se nesmí rovnat $n \equiv 12 \pmod{24}$ a $n \equiv 18 \pmod{24}$.*

Díky Větě 7 víme, že rozšířené hendikepové 2-pravidelné grafy jsou vždy tvořeny komponentami C_6. Což nám dává informaci i o tom, že pokud rozšířené hendikepové ohodnocení existuje na 2-pravidelném grafu, tak musí počet vrcholů být dělitelný šesti a navíc počet vrcholů nesmí být $n \equiv 12 \pmod{24}$ nebo $n \equiv 18 \pmod{24}$.

Velkým přínosem je Věta 8, která říká, že rozšířený hendikepový 2-pravidelný graf existuje, jestliže $n \equiv 0, 6 \pmod{24}$. Důkaz je veden konstruktivně a umíme takový 2-pravidelný graf s rozšířeným hendikepovým ohodnocením sestavit.

Věta 8 *[1] Rozšířené hendikepové ohodnocení pro 2-pravidelný graf G na n vrcholech existuje, jestliže $n \equiv 0, 6 \pmod{24}$, navíc struktura grafu je jednoznačně určena. Jedná se o $t \cdot C_6$.*

4.3 3-pravidelné grafy s rozšířeným hendikepovým ohodnocením

Matěj Krbeček ve své diplomové práci [1] zmiňuje v sekci odvození konstanty ℓ rozšířeného hendikepového ohodnocení, že rozšířené hendikepové ohodnocení neexistuje pro r-pravidelné grafy, kde r je liché číslo a současně n je liché číslo. Jedinou možností, pro které má smysl zkoumat rozšířené hendikepové ohodnocení na r-pravidelných grafech za podmínky, že r je liché číslo, je pouze na sudém počtu vrcholů grafu. Kde konstanta $\ell = \frac{(r-1)(n+2)}{2}$ je přirozené číslo, pro r liché větší než jedna a n sudé. Uvedeme zde pár vět o neexistenci 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením a nakonec zmíníme Větu 11 o existenci 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením kde n je násobkem čísla 8.

Věta 9 *Neexistuje 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na 4 vrcholech.*

Důkaz Tuto větu dokážeme sporem. Předpokládejme, že existuje 3-pravidelný graf na čtyřech vrcholech s rozšířeným hendikepovým ohodnocením. Nejprve vypočteme konstantu ℓ ze vztahu, který je uveden ve Větě 4 pro 3-pravidelný graf na čtyřech vrcholech s rozšířeným hendikepovým ohodnocením.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(3-1)(4+2)}{2} = \frac{2 \cdot 6}{2} = 6$$

V dalším kroku si sestavíme množinu přípustných labelů rozšířeného hendikepového ohodnocení.

$$y = \frac{(n+2)}{2} = \frac{(4+2)}{2} = \frac{6}{2} = 3$$
$$\left\{1, 2, \ldots, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\} = \{1, 2, 4, 5\}$$

Nyní víme, že množina labelů rozšířeného hendikepového ohodnocení pro 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na čtyřech vrcholech je $\{1, 2, 4, 5\}$. Označme ohodnocený vrchol labelem 1 jako v_1, a jeho tři sousední vrcholy jako u_1, u_2 a u_3. Potom podle Definice 16 rozšířeného hendikepového ohodnocení musí platit, že váha vrcholu v_1:

$$\sum_{u \in N(v_1)} f(u) = \ell + f(v_1)$$
$$f(u_1) + f(u_2) + f(u_3) = 6 + 1 = 7$$

Nejmenší hodnotu součtu tří labelů, ze zbývajících čísel, které je možné vytvořit, je

$$f(u_1) + f(u_2) + f(u_3) = 2 + 4 + 5 = 11,$$

což se nerovná číslu 7. Protože neexistuje žádná přípustná možnost tří vrcholů sousedních s vrcholem v_1 mající součet labelů 7, není možné vytvořit žádný 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na čtyřech vrcholech. □

Věta 10 *Neexistuje 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na 6 vrcholech.*

Důkaz Tuto větu opět dokážeme sporem. Předpokládejme, že existuje 3-pravidelný graf na šesti vrcholech s rozšířeným hendikepovým ohodnocením. Nejprve vypočteme konstantu ℓ ze vztahu, který je uveden ve Větě 4 pro 3-pravidelný graf na šesti vrcholech s rozšířeným hendikepovým ohodnocením.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(3-1)(6+2)}{2} = \frac{2 \cdot 8}{2} = 8$$

V dalším kroku si sestavíme množinu přípustných labelů rozšířeného hendikepového ohodnocení.

$$y = \frac{(n+2)}{2} = \frac{(6+2)}{2} = \frac{8}{2} = 4$$
$$\left\{1, 2, \ldots, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\} = \{1, 2, 3, 5, 6, 7\}$$

Nyní víme, že množina labelů rozšířeného hendikepového ohodnocení pro 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na čtyřech vrcholech je $\{1, 2, 3, 5, 6, 7\}$. Označme ohodnocený vrchol labelem 1 jako v_1, a jeho tři sousední vrcholy jako u_1, u_2 a u_3. Potom podle Definice 16 rozšířeného hendikepového ohodnocení musí platit, že váha vrcholu v_1:

$$\sum_{u \in N(v_1)} f(u) = \ell + f(v_1)$$
$$f(u_1) + f(u_2) + f(u_3) = 8 + 1 = 9$$

Nejmenší hodnotu součtu tří labelů, ze zbývajících čísel, které je možné vytvořit, je

$$f(u_1) + f(u_2) + f(u_3) = 2 + 3 + 5 = 10,$$

což se nerovná číslu 9. Protože neexistuje žádná přípustná možnost tří vrcholů sousedních s vrcholem v_1 mající součet labelů 9, není možné vytvořit žádný 3-pravidelný graf s rozšířeným hendikepovým ohodnocením na šesti vrcholech. □

Věta 11 *Rozšířené hendikepové ohodnocení 3-pravidelných grafů na n vrcholech existuje, jestliže platí*

$$n \equiv 0 \pmod 8.$$

Výsledný graf určíme jako tQ_3, kde $t = \frac{n}{8}$

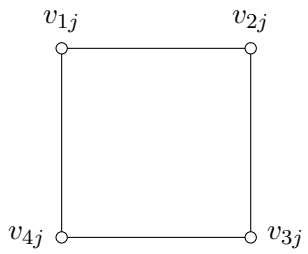
Důkaz Důkaz bude konstruktivní. Popíšeme konstrukci, jak sestrojit 3-pravidelný graf s rozšířeným hendikepovým ohodnocením s n vrcholy, kde n musí být $n \equiv 0 \pmod 8$.

Mějme tedy graf graf $G = tQ_3$, kde $t = \frac{n}{8}$. Využijeme zde hranově disjunktního rozkladu grafu hyperkrychle Q_3 na faktory Q'_3 a Q''_3. Připomeňme, že faktor Q'_3 tvoří dvě komponenty C_4. Faktor Q''_3 je tvořen čtyřmi komponentami kompletního grafu K_2. Označme symbolem G' a G'' rozklad grafu G na hranově disjunktní faktory, kde každou komponentu Q_3 v grafu G rozložíme na faktory Q'_3 a Q''_3.

$$\begin{aligned} G &= tQ_3 \\ G' &= tQ'_3 = 2tC_4 \\ G'' &= tQ''_3 = 4tK_2 \end{aligned}$$

Vidíme, že faktor G' je tvořen $2t$ komponentami cyklů C_4 a faktor G'' je tvořen $4t$ komponentami K_2. Komponenty C_4 ve faktoru G' budeme rozlišovat indexem j. Vrchol ve faktoru G' budeme značit v_{ij}, kde index i udává patřičný vrchol v dané komponentě j (viz Obrázek 11).

Uveďme patřičné rozsahy pro nově označené indexy $i \in \{1,2,3,4\}$ a $j \in \{1,2,\ldots,\frac{n}{4} = 2t\}$. Všimněme si, že pro $n \equiv 0 \pmod 8$ je $\frac{n}{4}$ je sudé číslo.



Obrázek 11: Pořadí označených vrcholů v j komponentě C_4 ve faktoru Q_3'

Poznámka 5 Připomeňme, že Q_3' je faktor Q_3. Jednou z vlastností faktoru je $V(Q_3) = V(Q_3')$, z toho vyplývá i informace, že $|V(Q_3)| = |V(Q_3')|$ Ověříme:

$$\begin{aligned}
&|V(Q_3)| = 8\\
&|V(Q_3')| = 2 \cdot |V(C_4)| = 2 \cdot 4 = 8\\
&|V(Q_3)| = |V(Q_3')|
\end{aligned}$$

Také i G' je faktor grafu G a platí $V(G) = V(G')$. Z této informace vyplývá i to, že $|V(G)| = |V(G')|$. Ověříme:

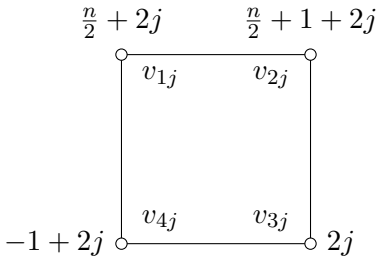
$$\begin{aligned}
&|V(G)| = t \cdot |V(Q_3)| = t \cdot 8 = \frac{n}{8} \cdot 8 = n\\
&|V(G')| = t \cdot |V(Q_3')| = 2t \cdot |V(C_4)| = \frac{2n}{8} \cdot 4 = n\\
&|V(G)| = |V(G')|
\end{aligned}$$

Definujeme zobrazení f takto:

$$f \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}. \tag{1}$$

$$f(v_{ij}) \quad = \quad \begin{cases} \frac{n}{2} + 2j & \text{pro} \quad i = 1,\\ \frac{n}{2} + 1 + 2j & \text{pro} \quad i = 2,\\ 2j & \text{pro} \quad i = 3,\\ -1 + 2j & \text{pro} \quad i = 4. \end{cases} \tag{2}$$

Toto zobrazení budeme nadále nazývat ohodnocením grafu G, protože $V(G) = V(G')$.
Nyní dokážeme bijektivnost zobrazení f. Čímž máme na mysli přiřazení, kde každému vrcholu přiřadíme právě jednu váhu, a navíc každá váha se bude vyskytovat právě jedenkrát. Na Obrázku 12 můžeme vidět ohodnocený cyklus C_4 v ohodnocení f.



Obrázek 12: Ohodnocené vrcholy v komponentě cyklu C_4 ve faktoru Q'_3

Označme množinu obrazů O_1 při zobrazení f, kde množinou vzorů jsou vrcholy v_{1j}:

$$O_1 = \left\{ f(v_{1j}) = \frac{n}{2} + 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{4}\right\}\right\} = \left\{\frac{n}{2}+2, \frac{n}{2}+4, \ldots, n-2, n\right\}$$

Označme množinu obrazů O_2 při zobrazení f, kde množinou vzorů jsou vrcholy v_{2j}:

$$O_2 = \left\{ f(v_{2j}) = \frac{n}{2} + 1 + 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{4}\right\}\right\} = \left\{\frac{n}{2}+3, \frac{n}{2}+5, \ldots, n-1, n+1\right\}$$

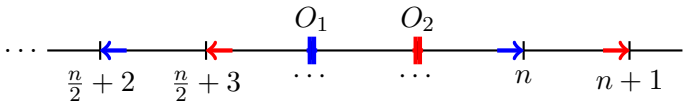
Označme množinu obrazů O_3 při zobrazení f, kde množinou vzorů jsou vrcholy v_{3j}:

$$O_3 = \left\{ f(v_{3j}) = 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{4}\right\}\right\} = \left\{2, 4, \ldots, \frac{n}{2}-2, \frac{n}{2}\right\}$$

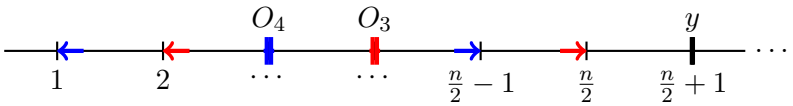
Označme množinu obrazů O_4 při zobrazení f, kde množinou vzorů jsou vrcholy v_{4j}:

$$O_4 = \left\{ f(v_{4j}) = -1 + 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{4}\right\}\right\} = \left\{1, 3, \ldots, \frac{n}{2}-3, \frac{n}{2}-1\right\}$$

Názorné zakreslení množin O_i můžeme vidět v Obrázcích 13 a 14



Obrázek 13: Zobrazení množiny obrazů vrcholů v_{1j} a v_{2j} při ohodnocení f



Obrázek 14: Zobrazení množiny obrazů vrcholů v_{3j} a v_{4j} při ohodnocení f

Pokud nyní sjednotíme všechny čtyři vypočtené množiny obrazů, dostaneme celou množinu

hodnot ohodnocení f.

$$O = \bigcup_{i=1}^{4} O_i = \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}$$

Všimneme si, že symbolem y v Obrázku 14, je označeno chybějící číslo v ohodnocení f, které má hodnotu $\frac{n}{2} + 1$. V Obrázku 13 jsou modrou barvou zobrazeny prvky množiny O_1, kde labely jsou vždy sudá čísla, a červenou barvou jsou zobrazeny prvky množiny O_2, kde labely jsou vždy lichá čísla.

Taktéž i v Obrázku 14 jsou modrou barvou zobrazeny prvky množiny O_4, kde labely jsou vždy lichá čísla, a červenou barvou jsou zaznačeny prvky množiny O_3, kde váhy jsou vždy sudá čísla. Navíc jsme ohodnotili každý vrchol právě jednou a můžeme vidět

$$O_m \bigcap O_n = \emptyset \quad \text{pro} \quad 1 \leq m < n \leq 4.$$

Dokázali jsme, že zobrazení f je bijekce. Nyní, když máme náš faktor G', očíslujme v něm vrcholy ve všech komponentách C_4 podle schématu v Obrázku 11 a vypočítejme váhy každého vrcholu při ohodnocení f (viz Obrázek 12). Množinu $J = \{1,\ 2,\ \ldots,\ \frac{n}{4} = 2t\}$ označující indexy komponent cyklů C_4 faktoru G', rozdělme na polovinu. Označme symbolem $L = \{1, 2, \ldots, \frac{n}{8}\}$ tuto množinu. Nyní vytvořme t dvojic $(l,\ m)$, tak že $l \in L = \{1, 2, \ldots, \frac{n}{8}\}$ a $m = (\frac{n}{4} - l + 1) \in J \setminus L$. Každá z těchto t dvojic tvoří samostatně labely faktoru Q_3'. Využijeme v této chvíli znalosti vztahu hranově disjunktního faktoru Q_3'' k faktoru Q_3' k doplnění hran na graf Q_3. Dále víme o grafu Q_3, že je komponentou grafu G.

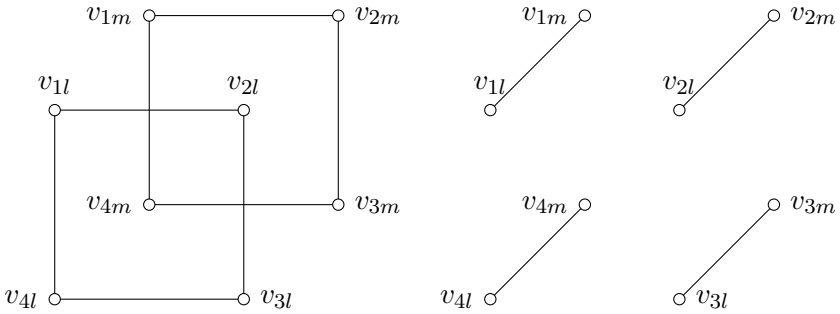
Poznámka 6 Připomeňme, že sjednocení hran faktorů Q_3' a Q_3'' tvoří hrany grafu Q_3.

$$\begin{aligned} V(Q_3) &= V(Q_3') = V(Q_3'') \\ Q_3 &= (V(Q_3),\ E(Q_3') \cup E(Q_3'')) \end{aligned}$$

Uveďme nyní předpis, pomocí kterého z jedné dvojice dvou cyklů C_4, které jsou označeny indexy odpovídající l a m, vytvoříme komponentu Q_3 v grafu G. Množinu hran komponenty l označme symbolem E_l, jedná se o hrany cyklu C_4. Podobně množinu hran komponenty m označme symbolem E_m, taktéž se jedná o hrany cyklu C_4. Množinu čtyř hran, které propojí odpovídající dva cykly C_4 o indexech l a m, označme E_{lm}, jedná se o faktor Q_3'' (viz Obrázek 15).

$$\begin{aligned} E_m &= \{v_{1m}v_{2m}, v_{2m}v_{3m}, v_{3m}v_{4m}, v_{4m}v_{1m}\} \\ E_l &= \{v_{1l}v_{2l}, v_{2l}v_{3l}, v_{3l}v_{4l}, v_{4l}v_{1l}\} \\ E_{l,m} &= \{v_{1l}v_{1m}, v_{2l}v_{2m}, v_{3l}v_{3m}, v_{4l}v_{4m}\} \end{aligned}$$

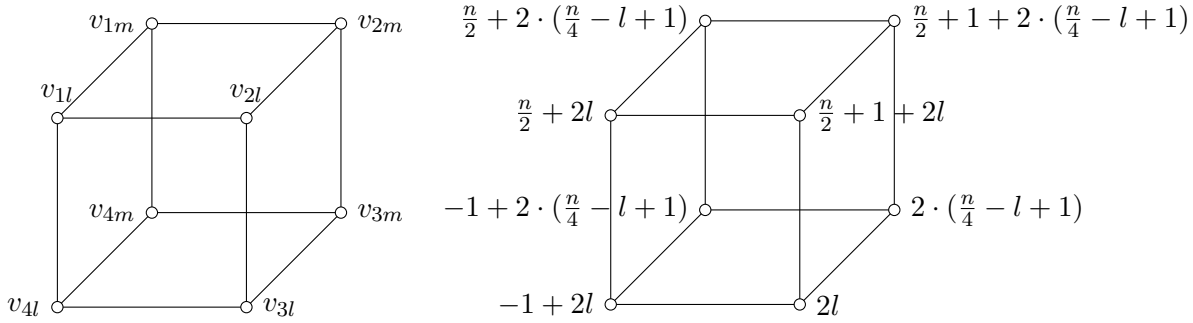
$$Q_3 = (\{v_{1l}, v_{2l}, v_{3l}, v_{4l}, v_{1m}, v_{2m}, v_{3m}, v_{4m}\},\ E_l \cup E_m \cup E_{l,m}) \tag{3}$$



(a) Komponenty l a m tvořící faktor Q'_3 (b) Faktor Q''_3

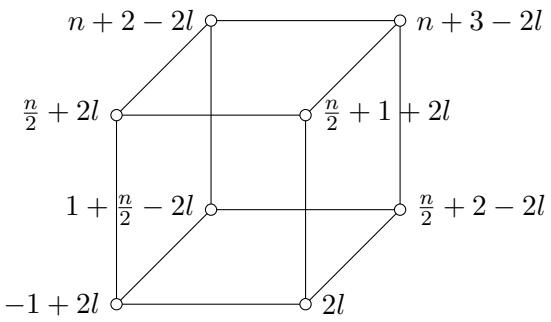
Obrázek 15: Faktory Q'_3 a Q''_3 při konstrukci komponenty Q_3 v grafu G

Díky rozdělení intervalu máme t různých dvojic faktorů Q'_3, proto jsme schopni vytvořit t různých komponent Q_3 pomocí předpisu 3. Výslednou komponentu Q_3 vidíme na Obrázku 16 spolu i s vypočtenými labely. Zkonstruovali jsme graf G, který je vytvořen pomocí t komponent Q_3, kde komponenta Q_3 je 3-pravidelný graf na osmi vrcholech. Tímto bychom měli konstrukci 3-pravidelného grafu hotovou.



(a) Komponenta Q_3 (b) Vypočtené labely v komponentě Q_3

Obrázek 16: Vypočtené labely v komponentě Q_3 grafu G



Obrázek 17: Vypočtené labely v komponentě Q_3 grafu G

Nyní ověříme, že váha každého vrcholu v komponentě Q_3 odpovídá rozšířenému hendikepovému ohodnocení. Dokážeme to pro každý vrchol v komponentě Q_3 a to za pomocí metody dvojího počítání. Konstantu ℓ vypočteme ze vztahu, který je uveden ve Větě 4.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(3-1)(n+2)}{2} = n+2$$

Nejprve vypočteme váhu vrcholu v_{1l} z definice rozšířeného hendikepového ohodnocení.

$$w_f(v_{1l}) = \ell + f(v_{1l}) = n+2+\frac{n}{2}+2l = \frac{3n}{2}+2+2l$$

Druhým výpočtem vypočítáme váhu v_{1l}, pomocí uspořádání labelů v komponentě Q_3. Tato váha je určena jako součet sousedních labelů vrcholu v_{1l} v ohodnocení f 1.

$$\begin{aligned} w_f(v_{1l}) &= \sum_{v \in N_G(v_{1l})} f(v) = f(v_{2l}) + f(v_{4l}) + f(v_{1m}) \\ &= \left(\frac{n}{2}+1+2l\right) + (-1+2l) + (n+2-2l) = \frac{3n}{2}+2+2l \end{aligned}$$

Vidíme, že oběma způsoby počítání jsme dostali stejný výsledek $\frac{3n}{2}+2+2l$. Ověřili jsme, že váha u vrcholů v_{1l} splňuje vlastnosti rozšířeného hendikepového ohodnocení. Ověříme, zda také ostatních sedm vrcholů v komponentě Q_3, splňuje vlastnosti rozšířeného hendikepového ohodnocení.

Podobně výpočet váhy pro vrchol v_{2l} je:

$$\begin{aligned} w_f(v_{2l}) &= \ell + f(v_{2l}) = n+2+\frac{n}{2}+1+2l = \frac{3n}{2}+3+2l \\ w_f(v_{2l}) &= \sum_{v \in N_G(v_{2l})} f(v) = f(v_{1l}) + f(v_{3l}) + f(v_{2m}) \\ &= \frac{n}{2}+2l+2l+n+3-2l = \frac{3n}{2}+3+2l \end{aligned}$$

Výpočet váhy pro vrchol v_{3l} je :

$$\begin{aligned} w_f(v_{3l}) &= \ell + f(v_{3l}) = n+2+2l \\ w_f(v_{3l}) &= \sum_{v \in N_G(v_{3l})} f(v) = f(v_{2l}) + f(v_{4l}) + f(v_{3m}) \\ &= \frac{n}{2}+1+2l-1+2l+\frac{n}{2}+2-2l = n+2+2l \end{aligned}$$

Výpočet váhy pro vrchol v_{4l} je:

$$\begin{aligned} w_f(v_{4l}) &= \ell + f(v_{4l}) = n + 2 - 1 + 2l = n + 1 + 2l \\ w_f(v_{4l}) &= \sum_{v \in N_G(v_{4l})} f(v) = f(v_{3l}) + f(v_{1l}) + f(v_{4m}) \\ &= 2l + \frac{n}{2} + 2l + 1\frac{n}{2} - 2l = n + 1 + 2l \end{aligned}$$

Výpočet váhy pro vrchol v_{1m} je:

$$\begin{aligned} w_f(v_{1m}) &= \ell + f(v_{1m}) = n + 2 + n + 2 - 2l = 2n + 4 - 2l \\ w_f(v_{1m}) &= \sum_{v \in N_G(v_{1m})} f(v) = f(v_{2m}) + f(v_{4m}) + f(v_{1l}) \\ &= n + 3 - 2l + 1 + \frac{n}{2} - 2l + \frac{n}{2} + 2l = 2n + 4 - 2l \end{aligned}$$

Výpočet váhy pro vrchol v_{2m} je:

$$\begin{aligned} w_f(v_{2m}) &= \ell + f(v_{2m}) = n + 2 + n + 3 - 2l = 2n + 5 - 2l \\ w_f(v_{2m}) &= \sum_{v \in N_G(v_{2m})} f(v) = f(v_{1m}) + f(v_{3m}) + f(v_{2l}) \\ &= n + 2 - 2l + \frac{n}{2} + 2 - 2l + \frac{n}{2} + 1 + 2l = 2n + 5 - 2l \end{aligned}$$

Výpočet váhy pro vrchol v_{3m} je:

$$\begin{aligned} w_f(v_{3m}) &= \ell + f(v_{3m}) = n + 2 + \frac{n}{2} + 2 - 2l = \frac{3n}{2} + 4 - 2l \\ w_f(v_{3m}) &= \sum_{v \in N_G(v_{3m})} f(v) = f(v_{2m}) + f(v_{4m}) + f(v_{3l}) \\ &= n + 3 - 2l + 1 + \frac{n}{2} - 2l + 2l = \frac{3n}{2} + 4 - 2l \end{aligned}$$

Výpočet váhy pro vrchol v_{4m} je:

$$\begin{aligned} w_f(v_{4m}) &= \ell + f(v_{4m}) = n + 2 + 1 + \frac{n}{2} - 2l = \frac{3n}{2} + 3 - 2l \\ w_f(v_{4m}) &= \sum_{v \in N_G(v_{4m})} f(v) = f(v_{3m}) + f(v_{1m}) + f(v_{4l}) \\ &= \frac{n}{2} + 2 - 2l + n + 2 - 2l - 1 + 2l = \frac{3n}{2} + 3 - 2l \end{aligned}$$

Ukázali jsme, že výpočet vah v komponentě Q_3 splňuje vlastnosti rozšířeného hendikepového ohodnocení pro každou různou dvojici komponent cyklů C_4 o indexech l a m. Pak celkový graf G, který je sestaven z t komponent Q_3, splňuje také rozšířené hendikepové ohodnocení na 3-pravidelném grafu na n vrcholech, kde $n \equiv 0 \pmod 8$. □

4.4 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením

Nejprve ukážeme, že pro malé počty vrcholů nemůže existovat 4-pravidelný graf s rozšířeným hendikepovým ohodnocením.

Věta 12 *Neexistuje 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 6 vrcholech.*

Důkaz Tuto větu dokážeme sporem. Předpokládejme, že existuje 4-pravidelný graf na šesti vrcholech s rozšířeným hendikepovým ohodnocením. Nejprve vypočteme konstantu ℓ ze vztahu, který je uveden ve Větě 4 pro 4-pravidelný graf na šesti vrcholech s rozšířeným hendikepovým ohodnocením.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(4-1)(6+2)}{2} = \frac{3 \cdot 8}{2} = 12$$

V dalším kroku si sestavíme množinu přípustných labelů rozšířeného hendikepového ohodnocení.

$$y = \frac{(n+2)}{2} = \frac{(6+2)}{2} = \frac{8}{2} = 4$$
$$\left\{1, 2, \ldots, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\} = \{1, 2, 3, 5, 6, 7\}$$

Nyní víme, že množina labelů rozšířeného hendikepového ohodnocení pro 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na šesti vrcholech je $\{1, 2, 3, 5, 6, 7\}$. Označme ohodnocený vrchol labelem 1 jako v_1, a jeho čtyři sousední vrcholy jako u_1, u_2, u_3 a u_4. Potom podle Definice 16 rozšířeného hendikepového ohodnocení musí platit, že váha vrcholu v_1:

$$\sum_{u \in N(v_1)} f(u) = \ell + f(v_1)$$
$$f(u_1) + f(u_2) + f(u_3) + f(u_4) = 12 + 1 = 13$$

Nejmenší hodnotu součtu čtyř labelů, ze zbývajících čísel, které je možné vytvořit, je

$$f(u_1) + f(u_2) + f(u_3) + f(u_4) = 2 + 3 + 5 + 6 = 16,$$

což se nerovná číslu 13. Protože neexistuje žádná přípustná možnost čtyř vrcholů sousedních s vrcholem v_1 mající součet labelů 13. Není tedy možné vytvořit žádný 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na šesti vrcholech. □

Věta 13 *Neexistuje 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 8 vrcholech.*

Důkaz Tuto větu dokážeme opět sporem. Předpokládejme, že takový 4-pravidelný graf G na osmi vrcholech s rozšířeným hendikepovým ohodnocením existuje. Nejprve vypočteme konstantu ℓ ze vztahu, který je uveden ve Větě 4 pro 4-pravidelný graf s rozšířeným hendikepovým ohod-

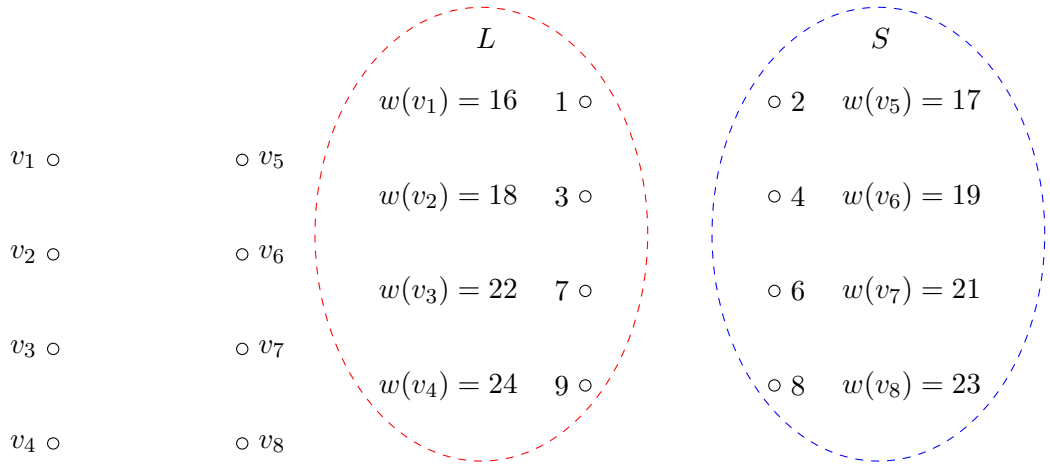
nocením na osmi vrcholech.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(4-1)(8+2)}{2} = \frac{3 \cdot 10}{2} = 15$$

V dalším kroku si sestavíme množinu přípustných labelů rozšířeného hendikepového ohodnocení.

$$y = \frac{(n+2)}{2} = \frac{8+2}{2} = 5$$
$$\left\{1, 2, \ldots, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\} = \{1, 2, 3, 4, 6, 7, 8, 9\}$$

Nyní víme, že množina labelů rozšířeného hendikepového ohodnocení pro 4-pravidelný graf na osmi vrcholech je $\{1, 2, 3, 4, 6, 7, 8, 9\}$. Nyní tuto množinu labelů rozšířeného hendikepového ohodnocení rozdělme na dvě množiny. V první množině L budou labely, které jsou liché $L = \{1, 3, 7, 9\}$. Ve druhé množině S budou labely, které jsou sudé $S = \{2, 4, 6, 8\}$. Zobrazme nyní hledaný graf G do dvou podmnožin, dle příslušných množin S a L. Uveďme ke každému vrcholu patřičnou váhu, kterou vypočteme ze vztahu z Definice 16 rozšířeného hendikepového ohodnocení $\forall v \in V(G) : w(v) = \ell + f(v)$, kde $f(v)$ je label vrcholu v a $w(v)$ je váha vrcholu v. Názorný příklad je zobrazen v Obrázku 18.



(a) Pořadí označených vrcholů v grafu G

(b) Přiřazené labely a vypočtené váhy v grafu G

Obrázek 18: Zobrazen graf G

Vidíme v Obrázku 18 (b), že váhy vrcholů mající label v množině L jsou sudá čísla. Existují pouze tři možnosti, jak získat sudý součet pomocí čtyř labelů s různou paritou.

1. sečteme čtyři sudé labely
2. sečteme čtyři liché labely
3. sečteme dva sudé labely a dva liché labely

První případ nemůže nastat, protože kdybychom sečetli všechny čtyři sudé labely v množině S, obdrželi bychom číslo $2 + 4 + 6 + 8 = 20$, které se nerovná žádné váze v 4-pravidelném grafu s rozšířeným hendikepovým ohodnocením na osmi vrcholech.

Druhý případ, při které bychom sečetli čtyři liché labely, taktéž nemůže nastat. Protože v množině L jsou právě čtyři liché labely a každý vrchol má právě čtyři sousední labely. Znamenalo by to, že mezi vrcholy, které mají liché labely buď existuje hrana do samotného vrcholu nebo vede multihrana mezi vrcholy. Obě tyto možnosti jsou v rozporu s Definicí 16, kde předpokládáme, že graf G je jednoduchý, ve kterém se multihrany ani smyčky nemohou vyskytnout.

Zbývá nám tedy poslední případ, která už nastat může a tedy každý vrchol mající lichý label má dvě hrany vedoucí do dvou vrcholů s lichými labely a zbylé dvě hrany vedou do dvou vrcholů se sudými labely. Tímto bychom měli množinu L lichých labelů probranou.

Teď se podíváme na množinu S sudých labelů. V Obrázku 18 vidíme, že váhy u vrcholů mající label v množině S jsou liché. Existují pouze dvě možnosti, jak získat lichý součet čtyř labelů s různou paritou.

1. sečteme tři labely liché a jeden label sudý
2. sečteme tři labely sudé a jeden label lichý

Podívejme se na vrchol v_5 v Obrázku 18. Vidíme, že jeho label je roven číslu dva a jeho váha je sedmnáct. Pokud by měl tři sousední vrcholy se sudými labely, jednoznačně by se jednalo o tyto vrcholy v_6, v_7 a v_8, mající labely $4, 6$ a 8. Pokud tyto tři labely sečteme, obdržíme číslo osmnáct, což je už více než váha sedmnáct, a proto vrchol v_5 může mít právě tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem.

Pokud se podíváme na vrchol v_8 v Obrázku 18, vidíme, že jeho label je roven číslu osm a váha je rovná číslu dvacet tři. Pokud by měl tři sousední vrcholy se sudými labely, jednoznačně by se jednalo o tyto vrcholy v_5, v_6 a v_7, mající labely $2, 4$ a 6. V součtu tyto labely dávají číslo dvanáct, proto aby váha vrcholu v_8 se rovnala číslu dvacet tři, musí být poslední lichý label roven číslu jedenáct. Takový lichý label s hodnotou jedenáct není v množině S, a proto vrchol v_8 s labelem osm může mít právě tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem.

Zbylé dva vrcholy v_6 a v_7 mající labely 4 a 6 v množině S, mohou mít tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem a nebo tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem. Tuto úlohu rozdělíme na čtyři možné případy a to tyhle

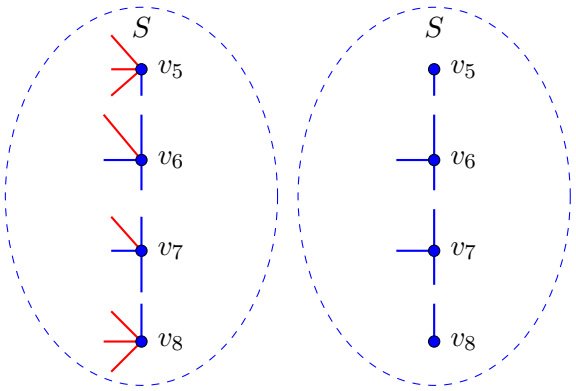
1. oba vrcholy v_6 a v_7 mají tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem,

2. oba vrcholy v_6 a v_7 mají tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem,

3. vrchol v_6 má tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem a vrchol v_7 má tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem,

4. vrchol v_6 má tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem a vrchol v_7 má tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem.

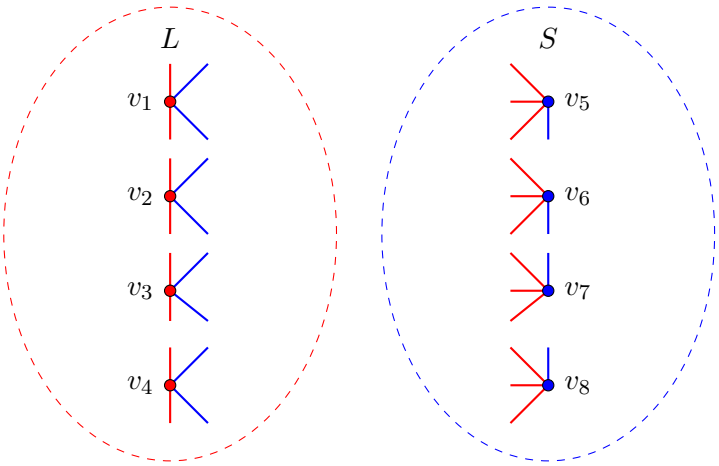
Nyní si rozebereme první případ, kdy vrcholy v_6 a v_7 mají tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem. Zobrazme si pouze vrcholy, které mají labely v množině S viz Obrázek 19 (a). Modrými čárami v Obrázku 19 (a) je symbolizovaná hrana do vrcholu se sudým labelem. Vrchol, který má sudý label je vybarven modrou barvou. Červenými čárami v Obrázku 19 je symbolizovaná hrana do vrcholu s lichým labelem. Nás ovšem bude zajímat pouze podgraf tvořený pouze modrými hranami viz Obrázek 19 (b). Vytvořme si stupňovou posloupnost tohoto podgrafu tvořeného modrými hranami z Obrázku 19 (b). Tato posloupnost je (1,3,3,1). Tuto stupňovou posloupnost sestupně uspořádejme a obdržíme tuto stupňovou posloupnost (3,3,1,1). Protože se jedná o podgraf a každý podgraf je také graf, můžeme použit k ověření Větu Havla-Hakimiho 1 a rozhodnout, zda se jedná o graf či nikoliv. Po aplikaci Věty Havla-Hakimiho 1 na stupňovou posloupnost (3,3,1,1) dostaneme posloupnost (2,0,0), která se dá ekvivalentně zapsat jako (2). Ze získaných výsledků vidíme, že žádný graf se stupňovou posloupností dva o jednom vrcholu neexistuje a proto ani původní stupňová posloupnost (1,3,3,1) není grafová. Nejedná se tak o graf, a proto tento případ nemůže nastat. Druhý



(a) Celý podgraf (b) Vybraný podgraf

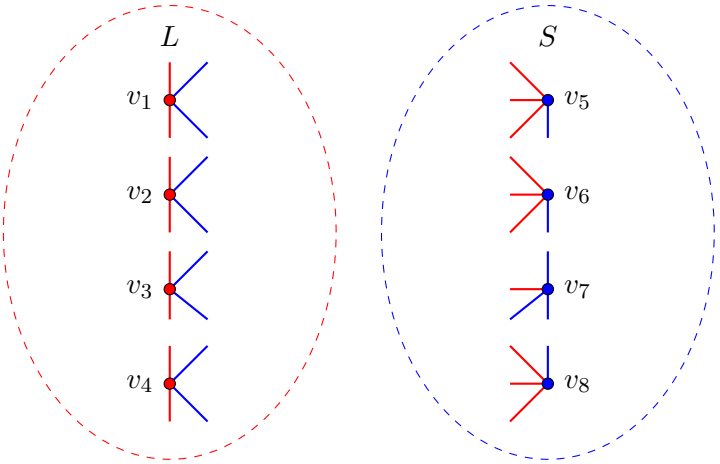
Obrázek 19: Podgraf grafu G, která má přiřazené sudé labely

případ je, když vrcholy v_6 a v_7 mají tři hrany, které vedou do vrcholů s lichými labely a jednu hranu, která vede do vrcholu se sudým labelem. Zobrazme si tuto situaci na Obrázku 20. Modré čáry znázorňují hrany spojené s vrcholy, které mají přiřazen sudý label a červené čáry znázorňují hrany spojené s vrcholy, které mají přiřazen lichý label. Modré vrcholy mají přiřazen sudý label a červené vrcholy mají přiřazen lichý label. Spočítejme počet hran mezi vrcholy (v_1, v_2, v_3, v_4) a (v_5, v_6, v_7, v_8). Z vrcholů v_1, v_2, v_3 a v_4 vede celkem 8 modrých hran do vrcholů v_5, v_6, v_7, v_8 a z vrcholů v_5, v_6, v_7, v_8 vede celkem 12 hran do vrcholů v_1, v_2, v_3, v_4. Tyto dvě čísla 8 a 12 se nerovnají a v jednoduchém grafu nemohou zůstat čtyři červené konce hran nepropojené. Taktéž i tento případ nemůže nastat.



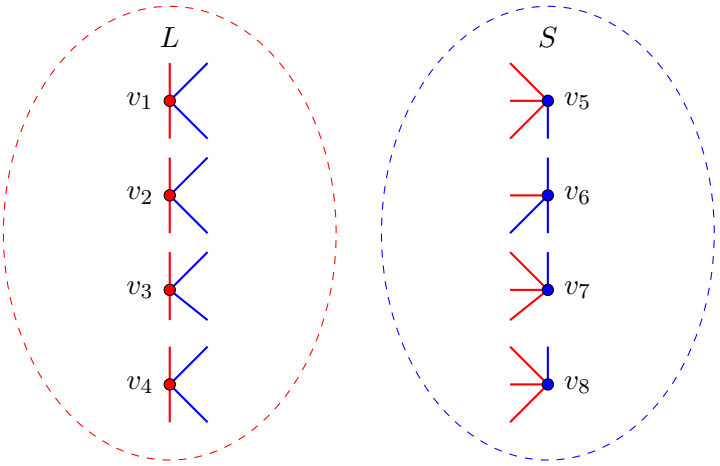
Obrázek 20: Zobrazen graf G varainta II

Třetí případ je, že vrchol v_6 má tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem a vrchol v_7 má tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem. Zobrazme si tuto situaci na Obrázku 21. Význam červených a modrých čar zůstává zachován, taktéž i význam modrých a červených vrcholů, které znázorňují vrchol se sudým nebo lichým labelem v Obrázku 21. Pokud opět sečteme počet modrých hran ve vrcholech v_1, v_2, v_3 a v_4 obdržíme číslo 8. Pokud nyní sečteme červené hrany ve vrcholech v_5, v_6, v_7 a v_8 obdržíme číslo 10. Taktéž i zde se čísla 8 a 10 nerovnají. Při propojení vrcholů v_1, v_2, v_3, v_4 s vrcholy v_5, v_6, v_7 a v_8 dva konce červených hran ve vrcholech v_5, v_6, v_7, v_8 budou nepropojené a není tedy možné je už nijak propojit, aby výsledný graf byl jednoduchý. Taktéž i tento případ nemůže nastat.



Obrázek 21: Zobrazen graf G varainta III

Poslední případ, který nám chybí ověřit, je, když vrchol v_6 má tři hrany vedoucí do vrcholů se sudými labely a jednu hranu vedoucí do vrcholu s lichým labelem a vrchol v_7 má tři hrany vedoucí do vrcholů s lichými labely a jednu hranu vedoucí do vrcholu se sudým labelem. Zobrazme si tuto situaci na Obrázku 22. Význam červených a modrých čar zůstává zachován, taktéž i význam modrých a červených vrcholů, které znázorňují vrchol se sudým nebo lichým labelem v Obrázku 22. Pokud opět sečteme počet modrých hran ve vrcholech v_1, v_2, v_3 a v_4 obdržíme číslo 8. Pokud sečteme i červené hrany ve vrcholech v_5, v_6, v_7 a v_8 obdržíme číslo 10. Dostáváme se do stejné situace jako v předchozí variantě, kdy čísla 8 a 10 se sobě nerovnají a ani tento případ nemůže nastat v jednoduchém grafu.



Obrázek 22: Zobrazen graf G varainta IV

Protože jsme ověřili všechny případy, tak jsme dokázali, že žádný takový 4-pravidelný graf s rozšířeným hendikepovým ohodnocením nemůže existovat. Každý případ vedl ke sporu. □

Nyní uveďme klíčovou větu o existenci rozšířeného hendikepového ohodnocení 4-pravidelných grafů. Na dále v této sekci grafem H rozumíme graf H ze sekce 2.4.

Věta 14 *Rozšířené hendikepové ohodnocení 4-pravidelných grafů na n vrcholech existuje, jestliže platí*

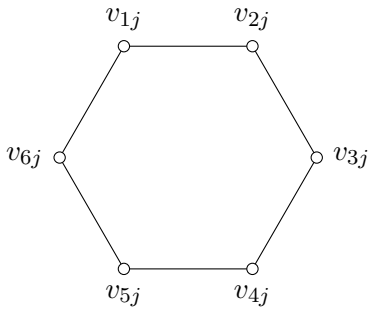
$$n \equiv 0 \pmod{12}.$$

Výsledný graf sestavíme jako tH, kde $t = \frac{n}{12}$

Důkaz Důkaz bude konstruktivní. Popíšeme konstrukci, jak sestrojit 4-pravidelný graf s rozšířeným hendikepovým ohodnocením, kde počet vrcholů n musí být $n \equiv 0 \pmod{12}$.
Mějme graf $G = tH$, kde $t = \frac{n}{12}$. Nejprve využijeme znalosti rozkladu grafu H ze sekce 2.4 a to na faktory H' a H''. Faktor H' tvoří právě dvě komponenty cyklů C_6. Faktor H'' je tvořen třemi komponentami cyklů C_4. Označme symbolem G' a G'' rozklad grafu G na dva hranově disjunktní faktory, kde komponenty H v grafu G rozložíme na faktor H' a H''.

$$\begin{aligned} G &= tH \\ G' &= tH' = 2tC_6 \\ G'' &= tH'' = 3tC_4 \end{aligned}$$

Vidíme, že faktor G' tvoří $2t$ komponent C_6 a faktor G'' je tvořen $3t$ komponentami C_4. Komponenty C_6 ve faktoru G' budeme rozlišovat indexem j. Vrchol ve faktoru G' budeme značit v_{ij}, kde index i udává patřičný vrchol v dané komponentě j (viz Obrázek 23).
Uveďme patřičné rozsahy pro nově označené indexy $i \in \{1,\ 2,\ \ldots,\ 6\}$ a $j \in \{1,\ 2,\ \ldots,\ \frac{n}{6} = 2t\}$. Připomeňme, že pro $n \equiv 0 \pmod{12}$ je $\frac{n}{6}$ sudé číslo.



Obrázek 23: Pořadí označených vrcholů v j-té komponentě C_6 ve faktoru G'

Poznámka 7 Připomeňme, že H' je faktor grafu H. Jednou z vlastností faktoru je $V(H) = V(H')$, z toho vyplývá i informace $|V(H)| = |V(H')|$. Ověříme:

$$\begin{aligned} |V(H)| &= 12 \\ |V(H')| &= 2 \cdot |V(C_6)| = 2 \cdot 6 = 12 \\ |V(H)| &= |V(H')| \end{aligned}$$

Také i G' je faktor grafu G a platí $V(G) = V(G')$. Z této informace vyplývá i to, že $|V(G)| = |V(G')|$. Ověříme:

$$\begin{aligned} |V(G)| &= t \cdot |V(H)| = t \cdot 12 = \frac{n}{12} \cdot 12 = n \\ |V(G')| &= t \cdot |V(H')| = 2t \cdot |V(C_6)| = \frac{2n}{12} \cdot 6 = n \\ |V(G)| &= |V(G')| \end{aligned}$$

Definujeme zobrazení f takto:

$$f \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}.$$

$$f(v_{ij}) \quad = \quad \begin{cases} \frac{n}{6} + 1 - j & \text{pro} \quad i = 1, \\ n + 3 - 2j & \text{pro} \quad i = 2, \\ \frac{2n}{3} + 2 - j & \text{pro} \quad i = 3, \\ \frac{n}{6} + j & \text{pro} \quad i = 4, \\ \frac{2n}{3} + 2j & \text{pro} \quad i = 5, \\ \frac{n}{3} + j & \text{pro} \quad i = 6. \end{cases}$$

Toto zobrazení budeme nadále nazývat ohodnocením grafu G, protože $V(G) = V(G')$. Nyní dokážeme, že zobrazení f je bijekce. Každému vrcholu přiřadíme právě jednu váhu a navíc každá váha se bude vyskytovat právě jedenkrát.

Označme množinu obrazů O_1 při zobrazení f, kde množinou vzorů jsou vrcholy v_{1j}

$$O_1 = \left\{f(v_{1j}) = \frac{n}{6} + 1 - j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{\frac{n}{6}, \frac{n}{6} - 1, \ldots, 2, 1\right\}.$$

Označme množinu obrazů O_2 při zobrazení f, kde množinou vzorů jsou vrcholy v_{2j}

$$O_2 = \left\{f(v_{2j}) = n + 3 - 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{n + 1, n - 1, \ldots, \frac{2n}{3} + 5, \frac{2n}{3} + 3\right\}.$$

Označme množinu obrazů O_3 při zobrazení f, kde množinou vzorů jsou vrcholy v_{3j}

$$O_3 = \left\{ f(v_{3j}) = \frac{2n}{3} + 2 - j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{\frac{2n}{3} + 1, \frac{2n}{3}, \ldots, \frac{n}{2} + 3, \frac{n}{2} + 2\right\}.$$

Označme množinu obrazů O_4 při zobrazení f, kde množinou vzorů jsou vrcholy v_{4j}

$$O_4 = \left\{ f(v_{4j}) = \frac{n}{6} + j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{\frac{n}{6} + 1, \frac{n}{6} + 2, \ldots, \frac{n}{3} - 1, \frac{n}{3}\right\}.$$

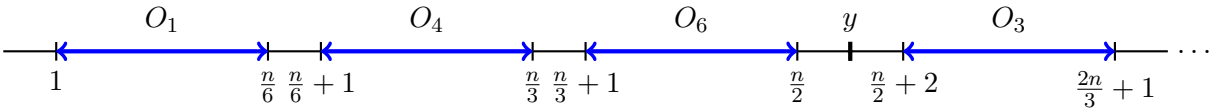
Označme množinu obrazů O_5 při zobrazení f, kde množinou vzorů jsou vrcholy v_{5j}

$$O_5 = \left\{ f(v_{5j}) = \frac{2n}{3} + 2j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{\frac{2n}{3} + 2, \frac{2n}{3} + 4, \ldots, n - 2, n\right\}.$$

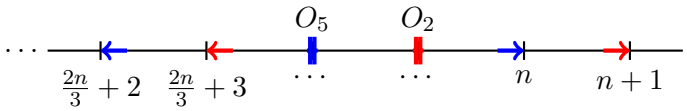
Označme množinu obrazů O_6 při zobrazení f, kde množinou vzorů jsou vrcholy v_{6j}

$$O_6 = \left\{ f(v_{6j}) = \frac{n}{3} + j \mid j \in \left\{1, 2, \ldots, \frac{n}{6}\right\}\right\} = \left\{\frac{n}{3} + 1, \frac{n}{3} + 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}\right\}.$$

Názorné zakreslení množin O_i můžeme vidět v Obrázcích 24 a 25.



Obrázek 24: Zobrazení množiny obrazů vrcholů v_{1j}, v_{4j}, v_{6j} a v_{3j} při ohodnocení f



Obrázek 25: Zobrazení množiny obrazů vrcholů v_{2j} a v_{5j} při ohodnocení f

Pokud nyní sjednotíme všech šest vypočtených množin obrazů, dostaneme celou množinu hodnot ohodnocení f.

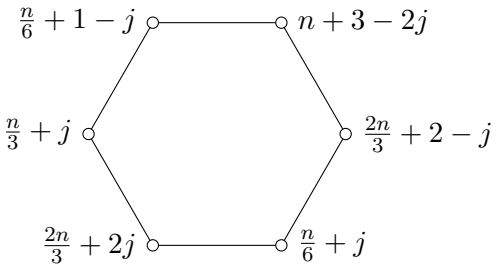
$$O = \bigcup_{i=1}^{6} O_i = \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n + 1\right\}$$

Všimneme si, že symbolem y v Obrázku 24, je označeno chybějící číslo v ohodnocení f, které má hodnotu $\frac{n}{2} + 1$. V Obrázku 25 jsou modrou barvou zobrazeny prvky množiny O_5, kde labely jsou vždy sudá čísla a červenou barvou jsou zaznačeny prvky množiny O_2, kde labely jsou vždy lichá čísla. Navíc jsme ohodnotili každý vrchol právě jednou a můžeme vidět

$$O_m \bigcap O_n = \emptyset \quad \text{pro} \quad 1 \leq m < n \leq 6.$$

Dokázali jsme, že zobrazení f je bijekce.

Nyní, když máme náš faktor G', očíslujme v něm vrcholy ve všech komponentách C_6 podle schématu v Obrázku 23 a vypočítejme labely každého vrcholu při ohodnocení f (viz Obrázek 26).



Obrázek 26: Pořadí vypočtených labelů u vrcholů j-té komponenty faktoru G'

Nyní množinu $\{1,\ 2,\ \ldots,\ \frac{n}{6}=2t\}$ označující komponenty C_6 faktoru G', rozdělme libovolně na t různých dvojic. Indexy k a l označme komponenty C_6 tvořící libovolnou dvojici, kde

$$k,l \in \{1,\ 2,\ \ldots,\ \frac{n}{6}=2t\} \wedge k \neq l.$$

Každá z těchto t dvojic tvoří samostatně labely vrcholu faktoru H'. Využijeme v této chvíli znalosti hranové disjunktnosti faktoru H'' k faktoru H' k doplnění hran do grafu H. Dále víme o grafu H, že je komponentou grafu G.

Poznámka 8 Připomeňme, že sjednocení hran faktorů H' a H'' tvoří graf H.

$$\begin{aligned} V(H) &= V(H') = V(H'') \\ H &= (V(G),\ E(H') \cup E(H'')) \end{aligned}$$

Hranovou množinu faktoru H $E_{k,l}$ definujeme pomocí sjednocení tří C_4 tak, jako ve faktoru H'' (viz Obrázek 27).

$$\begin{aligned} E_1 &= \{v_{1k}v_{1l}, v_{1k}v_{4l}, v_{4k}v_{1l}, v_{4k}v_{4l}\}, \\ E_2 &= \{v_{2k}v_{2l}, v_{2k}v_{5l}, v_{5k}v_{2l}, v_{5k}v_{5l}\}, \\ E_3 &= \{v_{3k}v_{3l}, v_{3k}v_{6l}, v_{6k}v_{3l}, v_{6k}v_{6l}\}, \\ E_{k,l} &= E_1 \cup E_2 \cup E_3 \end{aligned}$$

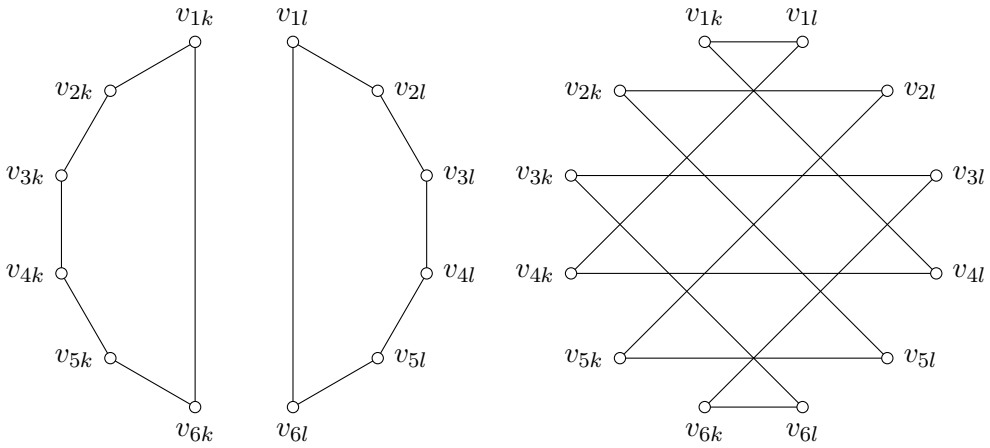
Hranovou množinu cyklu C_6 s indexem k označme E_k a hranovou množinu cyklu C_6 s indexem

l označme E_l.

$$\begin{aligned} E_k &= \{v_{1k}v_{2l}, v_{2k}v_{3l}, v_{3k}v_{4l}, v_{4k}v_{5l}, v_{5k}v_{6l}, v_{6k}v_{1l}\}, \\ E_l &= \{v_{1k}v_{2l}, v_{2l}v_{3l}, v_{3l}v_{4l}, v_{4l}v_{5l}, v_{5l}v_{6l}, v_{6l}v_{1l}\}. \end{aligned}$$

Uveďme nyní graf, pomocí kterého z jedné dvojice cyklů C_6 mající indexy k a l, vytvoříme komponentu H v grafu G.

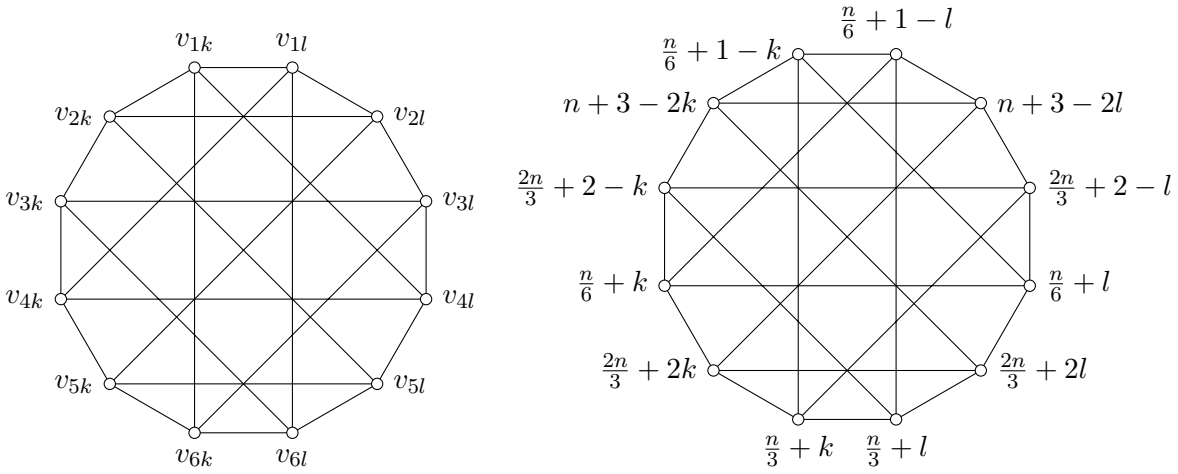
$$H = (\{v_{1k}, v_{2k}, v_{3k}, v_{4j}, v_{5j}, v_{6l}, v_{1l}, v_{2l}, v_{3l}, v_{4l}, v_{5l}, v_{6l}\},\ E_k \cup E_l \cup E_{k,l}) \tag{4}$$



(a) Komponenty k a l tvořící faktor H' (b) Faktor H''

Obrázek 27: Faktory H' a H'' při konstrukci komponenty H v grafu G

Díky rozdělení intervalu máme t různých dvojic faktorů H', jsme schopni vytvořit t různých komponent H pomocí předpisu 4. Výslednou komponentu H vidíme na Obrázku 28 spolu s vypočtenými labely. Zkonstruovali jsme graf G, který je vytvořen pomocí t komponent H, kde každá komponenta H je 4-pravidelný graf na 12 vrcholech. Tímto bychom měli konstrukci 4-pravidelného grafu hotovou.



(a) Komponenta H, která vznikla hranovým sjednocením hranově disjunktních faktorů H' a H''

(b) Vypočtené labely v komponentě H

Obrázek 28: Komponenta H grafu G

Nyní ověříme, že váha každého vrcholu v komponentě H odpovídá rozšířenému hendikepovému ohodnocení. Dokážeme to pro každý vrchol v komponentě H a to za pomocí metody dvojího počítání, podobně jako v důkazu Věty 11 Konstantu ℓ vypočteme ze vztahu, který je uveden ve Větě 4.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(4-1)(n+2)}{2} = \frac{3(n+2)}{2}$$

Nejprve vypočteme váhu vrcholu v_{1k} z definice rozšířeného hendikepového ohodnocení.

$$w_f(v_{1k}) = \ell + f(v_{1k}) = \frac{3(n+2)}{2} + \frac{n}{6} + 1 - k = \frac{5n}{3} + 4 - k$$

Druhým výpočtem vypočítáme váhu v_{1k} jako součet sousedních labelů vrcholu v_{1k} v komponentě H.

$$\begin{aligned} w_f(v_{1k}) &= \sum_{v \in N_G(v_{1k})} f(v) = f(v_{2k}) + f(v_{6k}) + f(v_{4l}) + f(v_{1l}) \\ &= (n+3-2k) + \left(\frac{n}{3}+k\right) + \left(\frac{n}{6}+l\right) + \left(\frac{n}{6}+1-l\right) = \frac{5n}{3} + 4 - k \end{aligned}$$

Vidíme, že oběma způsoby počítání jsme dostali stejný výsledek $\frac{5n}{3} + 4 - k$. Ověřili jsme, že váha u vrcholů v_{1k} splňuje vlastnosti rozšířeného hendikepového ohodnocení. Ověříme, zda také ostatních jedenáct vrcholů v komponentě H, splňuje vlastnosti rozšířeného hendikepového ohodnocení.

Podobně výpočet váhy pro vrchol v_{2k} je:

$$\begin{aligned} w_f(v_{2k}) &= \ell + f(v_{2k}) = \frac{3(n+2)}{2} + n + 3 - 2k = \frac{5n}{2} + 6 - 2k \\ w_f(v_{2k}) &= \sum_{v \in N_G(v_{2k})} f(v) = f(v_{1k}) + f(v_{3k}) + f(v_{5l}) + f(v_{2l}) \\ &= \frac{n}{6} + 1 - k + \frac{2n}{3} + 2 - k + \frac{2n}{3} + 2l + n + 3 - 2l = \frac{5n}{2} + 6 - 2k \end{aligned}$$

Výpočet váhy pro vrchol v_{3k} je:

$$\begin{aligned} w_f(v_{3k}) &= \ell + f(v_{3k}) = \frac{3(n+2)}{2} + \frac{2n}{3} + 2 - k = \frac{13n}{6} + 5 - k \\ w_f(v_{3k}) &= \sum_{v \in N_G(v_{3k})} f(v) = f(v_{2k}) + f(v_{4k}) + f(v_{6l}) + f(v_{3l}) \\ &= n + 3 - 2k + \frac{n}{6} + k + \frac{n}{3} + l + \frac{2n}{3} + 2 - l = \frac{13n}{6} + 5 - k \end{aligned}$$

Výpočet váhy pro vrchol v_{4k} je:

$$\begin{aligned} w_f(v_{4k}) &= \ell + f(v_{4k}) = \frac{3(n+2)}{2} + \frac{n}{6} + k = \frac{5n}{3} + 3 + k \\ w_f(v_{4k}) &= \sum_{v \in N_G(v_{4k})} f(v) = f(v_{3k}) + f(v_{5k}) + f(v_{1l}) + f(v_{4l}) \\ &= \frac{2n}{3} + 2 - k + \frac{2n}{3} + 2k + \frac{n}{6} + 1 - l + \frac{n}{6} + l = \frac{5n}{3} + 3 + k \end{aligned}$$

Výpočet váhy pro vrchol v_{5k} je:

$$\begin{aligned} w_f(v_{5k}) &= \ell + f(v_{5k}) = \frac{3(n+2)}{2} + \frac{2n}{3} + 2k = \frac{13n}{6} + 3 + 2k \\ w_f(v_{5k}) &= \sum_{v \in N_G(v_{5k})} f(v) = f(v_{4k}) + f(v_{6k}) + f(v_{2l}) + f(v_{5l}) \\ &= \frac{n}{6} + k + \frac{n}{3} + k + n + 3 - 2l + \frac{2n}{3} + 2l = \frac{13n}{6} + 3 + 2k \end{aligned}$$

Výpočet váhy pro vrchol v_{6k} je:

$$\begin{aligned} w_f(v_{6k}) &= \ell + f(v_{6k}) = \frac{3(n+2)}{2} + \frac{n}{3} + k = \frac{11n}{6} + 3 + k \\ w_f(v_{6k}) &= \sum_{v \in N_G(v_{6k})} f(v) = f(v_{1k}) + f(v_{5k}) + f(v_{3l}) + f(v_{6l}) \\ &= \frac{n}{6} + 1 - k + \frac{2n}{3} + 2k + \frac{2n}{3} + 2 - l + \frac{n}{3} + l = \frac{11n}{6} + 3 + k \end{aligned}$$

Výpočet váhy pro vrchol v_{1l} je:

$$\begin{aligned} w_f(v_{1l}) &= \ell + f(v_{1l}) = \frac{3(n+2)}{2} + \frac{n}{6} + 1 - l = \frac{5n}{3} + 4 - l \\ w_f(v_{1l}) &= \sum_{v \in N_G(v_{1l})} f(v) = f(v_{2l}) + f(v_{6l}) + f(v_{4k}) + f(v_{1k}) \\ &= n + 3 - 2l + \frac{n}{3} + l + \frac{n}{6} + k + \frac{n}{6} + 1 - k = \frac{5n}{3} + 4 - l \end{aligned}$$

Výpočet váhy pro vrchol v_{2l} je:

$$\begin{aligned} w_f(v_{2l}) &= \ell + f(v_{2l}) = \frac{3(n+2)}{2} + n + 3 - 2l = \frac{5n}{2} + 6 - 2l \\ w_f(v_{2l}) &= \sum_{v \in N_G(v_{2l})} f(v) = f(v_{1l}) + f(v_{3l}) + f(v_{5k}) + f(v_{2k}) \\ &= \frac{n}{6} + 1 - l + \frac{2n}{3} + 2 - l + \frac{2n}{3} + 2k + n + 3 - 2k = \frac{5n}{2} + 6 - 2l \end{aligned}$$

Výpočet váhy pro vrchol v_{3l} je:

$$\begin{aligned} w_f(v_{3l}) &= \ell + f(v_{3l}) = \frac{3(n+2)}{2} + \frac{2n}{3} + 2 - l = \frac{13n}{6} + 5 - l \\ w_f(v_{3l}) &= \sum_{v \in N_G(v_{3l})} f(v) = f(v_{2l}) + f(v_{4l}) + f(v_{6k}) + f(v_{3k}) \\ &= n + 3 - 2l + \frac{n}{6} + l + \frac{n}{3} + k + \frac{2n}{3} + 2 - k = \frac{13n}{6} + 5 - l \end{aligned}$$

Výpočet váhy pro vrchol v_{4l} je:

$$\begin{aligned} w_f(v_{4l}) &= \ell + f(v_{4l}) = \frac{3(n+2)}{2} + \frac{n}{6} + l = \frac{5n}{3} + 3 + l \\ w_f(v_{4l}) &= \sum_{v \in N_G(v_{4l})} f(v) = f(v_{3l}) + f(v_{5l}) + f(v_{1k}) + f(v_{4k}) \\ &= \frac{2n}{3} + 2 - l + \frac{2n}{3} + 2l + \frac{n}{6} + 1 - k + \frac{n}{6} + k = \frac{5n}{3} + 3 + l \end{aligned}$$

Výpočet váhy pro vrchol v_{5l} je:

$$w_f(v_{5l}) = \ell + f(v_{5l}) = \frac{3(n+2)}{2} + \frac{2n}{3} + 2l = \frac{13n}{6} + 3 + 2l$$
$$w_f(v_{5l}) = \sum_{v \in N_G(v_{5l})} f(v) = f(v_{4l}) + f(v_{6l}) + f(v_{2k}) + f(v_{5k})$$
$$= \frac{n}{6} + l + \frac{n}{3} + l + n + 3 - 2k + \frac{2n}{3} + 2k = \frac{13n}{6} + 3 + 2l$$

Výpočet váhy pro vrchol v_{6l} je:

$$w_f(v_{6l}) = \ell + f(v_{6l}) = \frac{3(n+2)}{2} + \frac{n}{3} + l = \frac{11n}{6} + 3 + l$$
$$w_f(v_{6l}) = \sum_{v \in N_G(v_{6l})} f(v) = f(v_{1l}) + f(v_{5l}) + f(v_{3k}) + f(v_{6k})$$
$$= \frac{n}{6} + 1 - l + \frac{2n}{3} + 2l + \frac{2n}{3} + 2 - k + \frac{n}{3} + k = \frac{11n}{6} + 3 + l$$

Ukázali jsme, že výpočet vah v komponentě H splňuje vlastnosti rozšířeného hendikepového ohodnocení pro každou utvořenou dvojici komponent C_6. Pak celkový graf G, který je sestaven z t komponent H, splňuje také podmínky rozšířeného hendikepového ohodnocení 4-pravidelných grafů na n vrcholech, kde $n \equiv 0 \pmod{12}$. □

Věta 15 *Mějme 4-pravidelný graf s n vrcholy, kde $n \equiv 0 \pmod{12}$, pak existuje alespoň $\frac{(2t)!}{2^t \cdot t!}$ různých ohodnocení stejného grafu G, jehož všechny komponenty budou isomorfní s grafem H, kde $t = \frac{n}{12}$.*

Důkaz Mějme tedy množinu $J = \{1, 2, \ldots, \frac{n}{6} = 2t\}$, kde prvky množiny J odpovídají indexům komponent C_6, které tvoří faktor G' z důkazu Věty 14. Z této množiny J, máme vytvořit t různých ohodnocených komponent H grafu G. K vytvoření každé komponenty H jsou zapotřebí dva cykly C_6, které tvoří faktor H' grafu H. Seřadíme všech $2t$ indexů cyklů C_6 do posloupnosti a dvojice cyklů C_6 propojíme hranami z faktoru H''. Vznikne nám t komponent H grafu G. Tímto způsobem dostaneme mnoho různých rozšířených hendikepových ohodnocení, ale ne všechny permutace vytvoří různé rozšířené hendikepové ohodnocení a na to se teď podíváme. Popíšeme kolika různými způsoby se dá zkonstruovat jedno libovolné isomorfní rozšířené hendikepové ohodnocení, kde všechny komponenty grafu G jsou isomorfní s grafem H. Nejdříve uspořádáme všech $2t$ indexů cyklů C_6 z množiny J libovolně do posloupnosti. V této posloupnosti vždy indexy cyklů C_6 na i-té a $(i+1)$. pozici použijeme ke konstrukci t grafů H přidáním hran z faktoru H'', dostaneme ohodnocený graf G který si označme G^*, kde $i \in \{1, 3, \ldots, 2t-3, 2t-1\}$. Tento vzniklý ohodnocený rozšířený hendikepový graf G^*, můžeme zapsat právě $2^t t!$ různými posloupnostmi. V posloupnosti máme t pozic, kde se může nacházet libovolný faktor H'. Na první pozici můžeme umístit t faktorů H' z grafů G^*. Na druhou pozici můžeme umístit jen $t-1$ faktorů H' z grafu G^*, protože jeden faktor H' jsme už umístili na první pozici, a proto nám zbývá k rozmístění

na 2. pozici v posloupnosti $t-1$ faktorů H' atd. Tento způsob vede na permutace $P(t)$, proto jsme schopni zkonstruovat celkem $t!$ různých posloupností sestavených z indexů C_6 množiny J, popisující stejné ohodnocení grafu G^*. Dále v každém z t faktorů H' nezáleží na pořadí indexů cyklů C_6 při konstrukci grafu H. To znamená, že každý faktor H', který je sestaven z indexů cyklů C_6 na i-té a $(i+1)$. pozici, je stejný ohodnocený faktor H', jako když bychom indexy cyklů C_6 nacházející se na i-té a $(i+1)$. pozici mezi sebou prohodili. V každé posloupnosti je t faktorů H' a můžeme se rozhodnout, zda u každého faktoru H' v posloupnosti ponecháme indexy nacházejících se na konkrétní i-té a $(i+1)$. pozici, nebo tyto indexy navzájem prohodíme. Jedná se pak o variaci s opakováním t prvků ze dvou možností a tedy jsme schopni z každé posloupnosti vytvořit 2^t navzájem různých posloupností popisující stejné ohodnocení grafu G^*. Protože máme $t!$ navzájem různých posloupností a z každé posloupnosti umíme vytvořit dalších 2^t různých posloupností, dostáváme počet různých posloupností jednoho ohodnocení grafu G^*, které je $P(t) \cdot V^*(t,2) = t! \cdot 2^t$. Proto abychom dostali počet různých hendikepových ohodnocení grafů G, podělíme proto počet všech možných posloupností sestavených z $2t$ indexů cyklů C_6, kterých je $P(2t) = 2t!$, počtem všech posloupností jak lze zapsat jedno libovolné ohodnocení grafu G^*.

$$\frac{P(2t)}{P(t) \cdot V^*(t,2)} = \frac{(2t)!}{t! \cdot 2^t}$$

□

4.5 Další konstrukce existence 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením

Díky počítačovému programu implementující Algoritmus 8, který je uveden v sekci 5, se nám podařilo nalézt i jiné 4-pravidelné grafy na 12 vrcholech. Z některých těchto grafů jsme odvodili i jiný předpis pro funkci f, než která je zmíněna v důkazu Věty 14. Tím jsme dokázali existenci jiné nekonečné třídy grafů kde n musí být násobek dvanácti. Uvedeme předpis pro jinou funkci f a ověříme, že také vznikne 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na n vrcholech, kde $n \equiv 0 \pmod{12}$. Formální důkaz by byl obdobný jako pro Větu 14. Ukážeme si jen nejdůležitější části pro nové ohodnocení.

Důkaz Označme symbolem f_2 novou funkci, kterou budeme nadále nazývat ohodnocením grafu G.

$$f_2 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\}.$$

$$f_2(v_{ij}) \quad = \quad \begin{cases} -2+3j & \text{pro} \quad i=1, \\ \frac{n}{2}+3-3j & \text{pro} \quad i=2, \\ n+5-6j & \text{pro} \quad i=3, \\ n+4-3j & \text{pro} \quad i=4, \\ \frac{n}{2}+3j & \text{pro} \quad i=5, \\ -4+6j & \text{pro} \quad i=6. \end{cases}$$

Naším úkolem bude dokázat, že zobrazení f_2 je bijekce z množiny vrcholů grafu G do množiny labelů.

Označme množinu obrazů O_1 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{1j}

$$O_1 = \left\{ f(v_{1j}) = -2+3j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \left\{ 1, 4, \ldots, \frac{n}{2}-5, \frac{n}{2}-2 \right\}.$$

Označme množinu obrazů O_2 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{2j}

$$O_2 = \left\{ f(v_{2j}) = \frac{n}{2}+3-3j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \left\{ \frac{n}{2}, \frac{n}{2}-3, \ldots, 6, 3 \right\}.$$

Označme množinu obrazů O_3 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{3j}

$$O_3 = \left\{ f(v_{3j}) = n+5-6j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \{ n-1, n-7, \ldots, 11, 5 \}.$$

Označme množinu obrazů O_4 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{4j}

$$O_4 = \left\{ f(v_{4j}) = n+4-3j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \left\{ n+1, n-2, \ldots, \frac{n}{2}+7, \frac{n}{2}+4 \right\}.$$

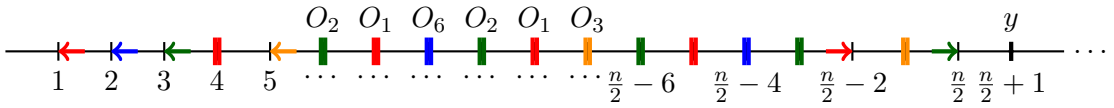
Označme množinu obrazů O_5 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{5j}

$$O_5 = \left\{ f(v_{5j}) = \frac{n}{2}+3j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \left\{ \frac{n}{2}+3, \frac{n}{2}+6, \ldots, n-3, n \right\}.$$

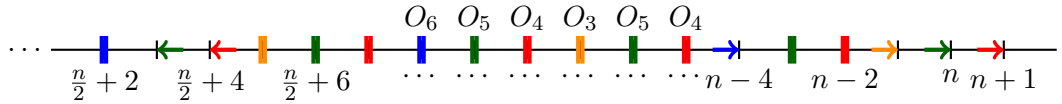
Označme množinu obrazů O_6 při zobrazení f_2, kde množinou vzorů jsou vrcholy v_{6j}

$$O_6 = \left\{ f(v_{6j}) = -4+6j \mid j \in \left\{ 1, 2, \ldots, \frac{n}{6} \right\} \right\} = \{ 2, 8, \ldots, n-10, n-4 \}.$$

Názorné zakreslení množin O_i můžeme vidět v Obrázcích 29 a 30.



Obrázek 29: Zobrazení množiny obrazů vrcholů v_{1j}, v_{2j}, v_{3j} a v_{6j} při ohodnocení f_2



Obrázek 30: Zobrazení množiny obrazů vrcholů v_{3j}, v_{4j}, v_{5j} a v_{6j} při ohodnocení f_2

Pokud nyní sjednotíme všech šest vypočtených množin obrazů, dostaneme celou množinu hodnot ohodnocení f_2.

$$O = \bigcup_{i=1}^{6} O_i = \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}$$

Všimneme si, že symbolem y v Obrázku 29, které má hodnotu $\frac{n}{2} + 1$ je označeno chybějící číslo v ohodnocení f_2. V Obrázku 29 jsou červenou barvou znázorněny prvky množiny O_1, modrou barvou jsou znázorněny prvky množiny O_6, zelenou barvou jsou znázorněny prvky množiny O_2 a oranžovou barvou jsou znázorněny prvky množiny O_3. V Obrázku 30 jsou modrou barvou znázorněny prvky množiny O_6, zelenou barvou jsou znázorněny prvky množiny O_5, červenou barvou jsou znázorněny prvky množiny O_4 a oranžovou barvou jsou znázorněny prvky množiny O_3. Navíc jsme ohodnotili každý vrchol právě jednou a můžeme vidět:

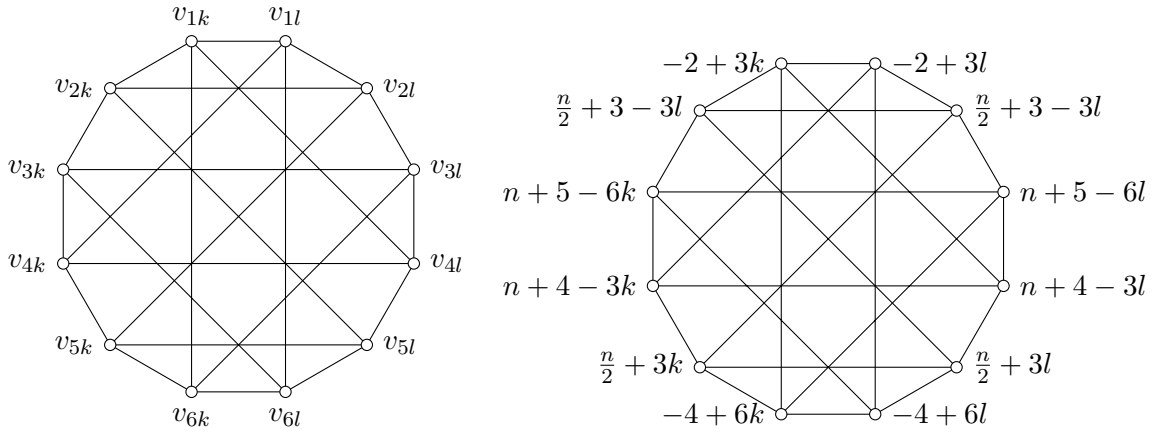
$$O_m \bigcap O_n = \emptyset \quad \text{pro} \quad 1 \leq m < n \leq 6$$

Dokázali jsme, že zobrazení f_2 je bijekce.

Nyní přiřadíme vrcholům komponent C_6 labely. Množinu komponent C_6 $\{1, 2, \ldots, \frac{n}{6} = 2t\}$ opět rozdělíme na t různých dvojic. Indexy k, l označme komponenty C_6 tvořící libovolnou dvojici, kde

$$k, l \quad \in \quad \left\{1, 2, \ldots, \frac{n}{6} = 2t\right\} \wedge k \neq l$$

Každá dvojice k a l tvoří faktor H'. Využijeme i zde znalosti hranově disjunktního faktoru H'' k doplnění hran na graf H, který tvoří komponentu grafu G. K doplnění hran využijeme stejný předpis 4 jako v důkazu Věty 14. Pro názornost uvádíme Obrázek 31.



(a) Komponenta H, která vznikla hranovým sjednocením hranově disjunktních faktorů H' a H''

(b) Vypočtené labely v komponentě H

Obrázek 31: Komponenta H při ohodnocení f_2

Ověříme, zda komponenta H splňuje vlastnosti rozšířeného hendikepového ohodnocení. Konstantu ℓ vypočteme ze vztahu, který je uveden ve Větě 4.

$$\ell = \frac{(r-1)(n+2)}{2} = \frac{(4-1)(n+2)}{2} = \frac{3(n+2)}{2}$$

Opět využijeme metodu dvojího počítání. Nejprve vypočteme váhu vrcholu v_{1k} z definice rozšířeného hendikepového ohodnocení.

$$w_{f_2}(v_{1k}) = \ell + f_2(v_{1k}) = \frac{3(n+2)}{2} - 2 + 3k = \frac{3n}{2} + 1 + 3k$$

Druhým výpočtem vypočítáme váhu v_{1k}, pomocí uspořádání labelů v komponentě H. Tato váha je určena jako součet sousedních labelů vrcholu v_{1k}.

$$\begin{aligned} w_{f_2}(v_{1k}) &= \sum_{v \in N_G(v_{1k})} f_2(v) = f_2(v_{2k}) + f_2(v_{6k}) + f_2(v_{4l}) + f_2(v_{1l}) \\ &= \frac{n}{2} + 3 - 3k - 4 + 6k + n + 4 - 3l - 2 + 3l = \frac{3n}{2} + 1 + 3k \end{aligned}$$

Vidíme, že oběma způsoby počítání jsme dostali stejný výsledek $\frac{3n}{2} + 1 + 3k$. Ověřili jsme, že váha u vrcholů v_{1k} splňuje vlastnosti rozšířeného hendikepového ohodnocení. Ověříme, zda také zbylých jedenáct vrcholů v komponentě H, splňuje vlastnosti rozšířeného hendikepového ohodnocení.

Podobně výpočet váhy pro vrchol v_{2k} je:

$$\begin{aligned} w_{f_2}(v_{2k}) &= \ell + f_2(v_{2k}) = \frac{3(n+2)}{2} + \frac{n}{2} + 3 - 3k = 2n + 6 - 3k \\ w_{f_2}(v_{2k}) &= \sum_{v \in N_G(v_{2k})} f_2(v) = f_2(v_{1k}) + f_2(v_{3k}) + f_2(v_{5l}) + f_2(v_{2l}) \\ &= -2 + 3k + n + 5 - 6k + \frac{n}{2} + 3l + \frac{n}{2} + 3 - 3l = 2n + 6 - 3k \end{aligned}$$

Výpočet váhy pro vrchol v_{3k} je:

$$\begin{aligned} w_{f_2}(v_{3k}) &= \ell + f_2(v_{3k}) = \frac{3(n+2)}{2} + n + 5 - 6k = \frac{5n}{2} + 8 - 6k \\ w_{f_2}(v_{3k}) &= \sum_{v \in N_G(v_{3k})} f_2(v) = f_2(v_{2k}) + f_2(v_{4k}) + f_2(v_{6l}) + f_2(v_{3l}) \\ &= \frac{n}{2} + 3 - 3k + n + 4 - 3k - 4 + 6l + n + 5 - 6l = \frac{5n}{2} + 8 - 6k \end{aligned}$$

Výpočet váhy pro vrchol v_{4k} je:

$$\begin{aligned} w_{f_2}(v_{4k}) &= \ell + f_2(v_{4k}) = \frac{3(n+2)}{2} + n + 4 - 3k = \frac{5n}{2} + 7 - 3k \\ w_{f_2}(v_{4k}) &= \sum_{v \in N_G(v_{4k})} f_2(v) = f_2(v_{3k}) + f_2(v_{5k}) + f_2(v_{1l}) + f_2(v_{4l}) \\ &= n + 5 - 6k + \frac{n}{2} + 3k - 2 + 3l + n + 4 - 3l = \frac{5n}{2} + 7 - 3k \end{aligned}$$

Výpočet váhy pro vrchol v_{5k} je:

$$\begin{aligned} w_{f_2}(v_{5k}) &= \ell + f_2(v_{5k}) = \frac{3(n+2)}{2} + \frac{n}{2} + 3k = 2n + 3 + 3k \\ w_{f_2}(v_{5k}) &= \sum_{v \in N_G(v_{5k})} f_2(v) = f_2(v_{4k}) + f_2(v_{6k}) + f_2(v_{2l}) + f_2(v_{5l}) \\ &= n + 4 - 3k - 4 + 6k + \frac{n}{2} + 3 - 3l + \frac{n}{2} + 3l = 2n + 3 + 3k \end{aligned}$$

Výpočet váhy pro vrchol v_{6k} je:

$$\begin{aligned} w_{f_2}(v_{6k}) &= \ell + f_2(v_{6k}) = \frac{3(n+2)}{2} - 4 + 6k = \frac{3n}{2} - 1 + 6k \\ w_{f_2}(v_{6k}) &= \sum_{v \in N_G(v_{6k})} f_2(v) = f_2(v_{1k}) + f_2(v_{5k}) + f_2(v_{3l}) + f_2(v_{6l}) \\ &= -2 + 3k + \frac{n}{2} + 3k + n + 5 - 6l - 4 + 6l = \frac{3n}{2} - 1 + 6k \end{aligned}$$

Výpočet váhy pro vrchol v_{1l} je:

$$\begin{aligned} w_{f_2}(v_{1l}) &= \ell + f_2(v_{1l}) = \frac{3(n+2)}{2} - 2 + 3l = \frac{3n}{2} + 1 + 3l \\ w_{f_2}(v_{1l}) &= \sum_{v \in N_G(v_{1l})} f_2(v) = f_2(v_{2l}) + f_2(v_{6l}) + f_2(v_{4k}) + f_2(v_{1k}) \\ &= \frac{n}{2} + 3 - 3l - 4 + 6l + n + 4 - 3k - 2 + 3k = \frac{3n}{2} + 1 + 3l \end{aligned}$$

Výpočet váhy pro vrchol v_{2l} je:

$$\begin{aligned} w_{f_2}(v_{2l}) &= \ell + f_2(v_{2l}) = \frac{3(n+2)}{2} + \frac{n}{2} + 3 - 3l = 2n + 6 - 3l \\ w_{f_2}(v_{2l}) &= \sum_{v \in N_G(v_{2l})} f_2(v) = f_2(v_{1l}) + f_2(v_{3l}) + f_2(v_{5k}) + f_2(v_{2k}) \\ &= -2 + 3l + n + 5 - 6l + \frac{n}{2} + 3k + \frac{n}{2} + 3 - 3k = 2n + 6 - 3l \end{aligned}$$

Výpočet váhy pro vrchol v_{3l} je:

$$\begin{aligned} w_{f_2}(v_{3l}) &= \ell + f_2(v_{3l}) = \frac{3(n+2)}{2} + n + 5 - 6l = \frac{5n}{2} + 8 - 6l \\ w_{f_2}(v_{3l}) &= \sum_{v \in N_G(v_{3l})} f_2(v) = f_2(v_{2l}) + f_2(v_{4l}) + f_2(v_{6k}) + f_2(v_{3k}) \\ &= \frac{n}{2} + 3 - 3l + n + 4 - 3l - 4 + 6k + n + 5 - 6k = \frac{5n}{2} + 8 - 6l \end{aligned}$$

Výpočet váhy pro vrchol v_{4l} je:

$$\begin{aligned} w_{f_2}(v_{4l}) &= \ell + f_2(v_{4l}) = \frac{3(n+2)}{2} + n + 4 - 3l = \frac{5n}{2} + 7 - 3l \\ w_{f_2}(v_{4l}) &= \sum_{v \in N_G(v_{4l})} f_2(v) = f_2(v_{3l}) + f_2(v_{5l}) + f_2(v_{1k}) + f_2(v_{4k}) \\ &= n + 5 - 6l + \frac{n}{2} + 3l - 2 + 3k + n + 4 - 3k = \frac{5n}{2} + 7 - 3l \end{aligned}$$

Výpočet váhy pro vrchol v_{5l} je:

$$\begin{aligned} w_{f_2}(v_{5l}) &= \ell + f(v_{5l}) = \frac{3(n+2)}{2} + \frac{n}{2} + 3l = 2n + 3 + 3l \\ w_{f_2}(v_{5l}) &= \sum_{v \in N_G(v_{5l})} f_2(v) = f_2(v_{4l}) + f_2(v_{6l}) + f_2(v_{2k}) + f_2(v_{5k}) \\ &= n + 4 - 3l - 4 + 6l + \frac{n}{2} + 3 - 3k + \frac{n}{2} + 3k = 2n + 3 + 3l \end{aligned}$$

Výpočet váhy pro vrchol v_{6l} je:

$$\begin{aligned} w_{f_2}(v_{6l}) &= \ell + f(v_{6l}) = \frac{3(n+2)}{2} - 4 + 6l = \frac{3n}{2} - 1 + 6l \\ w_{f_2}(v_{6l}) &= \sum_{v \in N_G(v_{6l})} f_2(v) = f_2(v_{1l}) + f_2(v_{5l}) + f_2(v_{3k}) + f_2(v_{6k}) \\ &= -2 + 3l + \frac{n}{2} + 3l + n + 5 - 6k - 4 + 6k = \frac{3n}{2} - 1 + 6l \end{aligned}$$

Ukázali jsme, že výpočet vah v komponentě H splňuje vlastnosti rozšířeného hendikepového ohodnocení, pro různou dvojici komponent C_6 k a l. Pak celkový graf G, který je tvořen z t komponent H, splňuje vlastnosti rozšířeného hendikepového ohodnocení na 4-pravidelném grafu na n vrcholech, kde $n \equiv 0 \pmod{12}$. □

4.5.1 Dalších šest nalezených různých předpisů pro ohodnocovací funkci f

Během práce se nám podařilo nalézt dalších šest různých předpisů pro ohodnocovací funkci f. Nově nalezené předpisy funkcí f budeme značit f_3, f_4, f_5, f_6, f_7 a f_8. Ke každé zmíněné funkci f pouze uvedeme předpis, důkaz již ukazovat nebudeme, byl by obdobný jako v předchozí sekci.

$$f_3 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}.$$

$$f_3(v_{ij}) \quad = \quad \begin{cases} \frac{n}{3} + 1 - 2j & \text{pro} \quad i = 1, \\ n + 2 - j & \text{pro} \quad i = 2, \\ \frac{2n}{3} + 1 + j & \text{pro} \quad i = 3, \\ 2j & \text{pro} \quad i = 4, \\ \frac{n}{2} + 1 + j & \text{pro} \quad i = 5, \\ \frac{n}{2} + 1 - j & \text{pro} \quad i = 6. \end{cases}$$

$$f_4 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}.$$

$$f_4(v_{ij}) \quad = \quad \begin{cases} n + 4 - 3j & \text{pro} \quad i = 1, \\ n + 6 - 6j & \text{pro} \quad i = 2, \\ \frac{n}{2} + 2 - 3j & \text{pro} \quad i = 3, \\ -2 + 3j & \text{pro} \quad i = 4, \\ -3 + 6j & \text{pro} \quad i = 5, \\ \frac{n}{2} - 1 + 3j & \text{pro} \quad i = 6. \end{cases}$$

$$f_5 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}.$$

$$f_5(v_{ij}) \quad = \quad \begin{cases} \frac{n}{3} + j & \text{pro} \quad i = 1, \\ \frac{2n}{3} - j + 2 & \text{pro} \quad i = 2, \\ n + 3 - 2j & \text{pro} \quad i = 3, \\ \frac{n}{6} + 1 - j & \text{pro} \quad i = 4, \\ \frac{n}{6} + j & \text{pro} \quad i = 5, \\ \frac{2n}{3} + 2j & \text{pro} \quad i = 6. \end{cases}$$

$$f_6 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2} - 1, \frac{n}{2}, \frac{n}{2} + 2, \frac{n}{2} + 3, \ldots, n, n+1\right\}.$$

$$f_6(v_{ij}) \quad = \quad \begin{cases} \frac{n}{3} + 1 - 2j & \text{pro} \quad i = 1, \\ n + 2 - j & \text{pro} \quad i = 2, \\ \frac{n}{2} + 1 + j & \text{pro} \quad i = 3, \\ 2j & \text{pro} \quad i = 4, \\ \frac{2n}{3} + 1 + j & \text{pro} \quad i = 5, \\ \frac{n}{2} + 1 - j & \text{pro} \quad i = 6. \end{cases}$$

$$f_7 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\}.$$

$$f_7(v_{ij}) \quad = \quad \begin{cases} -2+3j & \text{pro} \quad i=1, \\ -4+6j & \text{pro} \quad i=2, \\ \frac{n}{2}+3j & \text{pro} \quad i=3, \\ \frac{n}{2}+3-3j & \text{pro} \quad i=4, \\ n+5-6j & \text{pro} \quad i=5, \\ n+4-3j & \text{pro} \quad i=6. \end{cases}$$

$$f_8 \quad : \quad V(G') \to \left\{1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}, \frac{n}{2}+2, \frac{n}{2}+3, \ldots, n, n+1\right\}.$$

$$f_8(v_{ij}) \quad = \quad \begin{cases} n+4-3j & \text{pro} \quad i=1, \\ -2+3j & \text{pro} \quad i=2, \\ -3+6j & \text{pro} \quad i=3, \\ \frac{n}{2}-1+3j & \text{pro} \quad i=4, \\ \frac{n}{2}+2-3j & \text{pro} \quad i=5, \\ n+6-6j & \text{pro} \quad i=6. \end{cases}$$

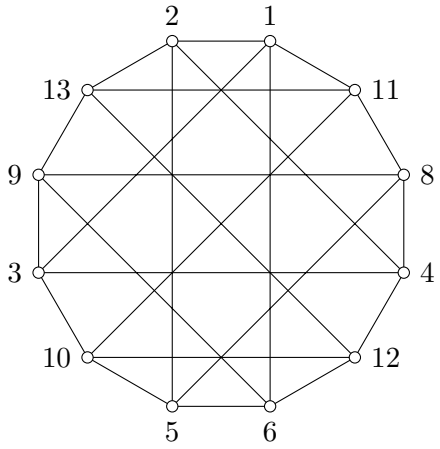
Porovnejme např. ohodnocení f a f_2. V ohodnocení f se labely u vrcholů v_{1j} a v_{1j+1} v komponentách j a $j+1$ liší o 1 a u vrcholů v_{2j} a v_{2j+1} se labely liší o 2. Oproti tomu v ohodnocení f_2 se labely u vrcholů v_{1j} a v_{1j+1} v komponentách j a $j+1$ liší o 3. Těmto číselným změnám dvou vrcholů v komponentách j a $j+1$ budeme říkat diference. Uvedený přehled diferencí u nalezených ohodnocení f, f_2, f_3, f_4, f_5, f_6, f_7 a f_8 shrnuje Tabulka 2. Tyto různá ohodnocení mohou pomoci v budoucnu při sestavování různých ohodnocení větších nebo hustších grafů.

funkce f	diference ±1 a ±2	diference ±3 a ±6
f	Ano	Ne
f_2	Ne	Ano
f_3	Ano	Ne
f_4	Ne	Ano
f_5	Ano	Ne
f_6	Ano	Ne
f_7	Ne	Ano
f_8	Ne	Ano

Tabulka 2: Diference funkcí f

Příklad 1

Protože je důkaz Věty 14 konstruktivní, můžeme sestrojit 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením na n vrcholech, kde $n \equiv 0 \pmod{12}$. Ukážeme si výsledný 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na dvanácti a dvaceti čtyřech vrcholech, které vznikly za pomocí konstrukce uvedené v důkazu Věty 14. Pomocí Věty 15 jsme schopni vypočítat počet různých ohodnocení grafů. Pro rozšířené hendikepové ohodnocení 4-pravidelných grafů. Na dvanácti vrcholech nám vyjde jedna možnost, jak sestavit tento graf ze dvou komponent C_6. Výsledný ohodnocený graf je zobrazen na Obrázku 32.

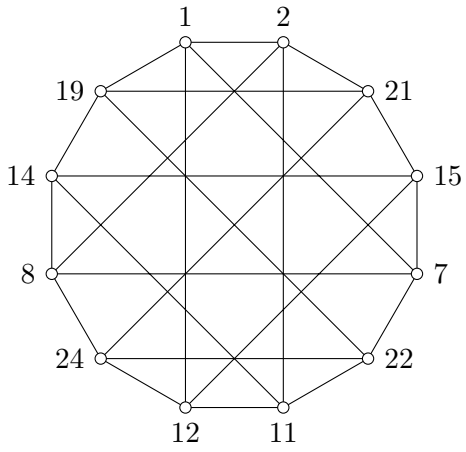


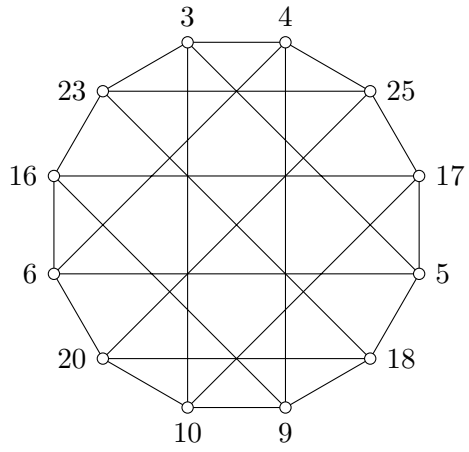
Obrázek 32: 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 12 vrcholech

Pro rozšířený hendikepový 4-pravidelný graf na dvaceti čtyřech vrcholech, kde parametr t je roven $t = \frac{n}{12} = \frac{24}{12} = 2$, nám podle Věty 15 vyjde, že existují alespoň tři možnosti, jak vytvořit různé 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením kde všechny komponenty jsou isomorfní s grafem H.

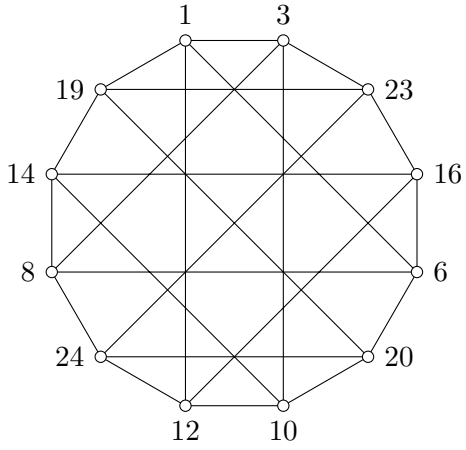
$$\frac{(2t)!}{2^t \cdot t!} = \frac{4!}{2^2 \cdot 2!} = \frac{4 \cdot 3 \cdot 2!}{4 \cdot 2!} = 3$$

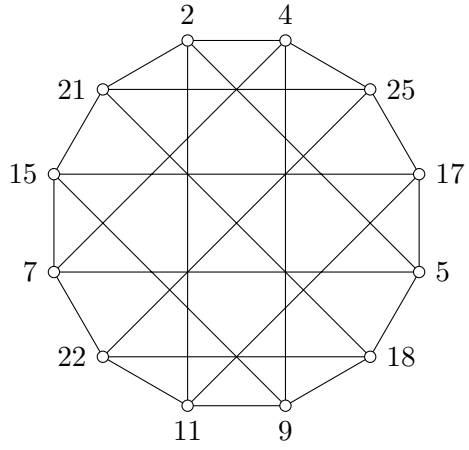
Všechny tři možnosti jsou zobrazeny v Obrázku 33.



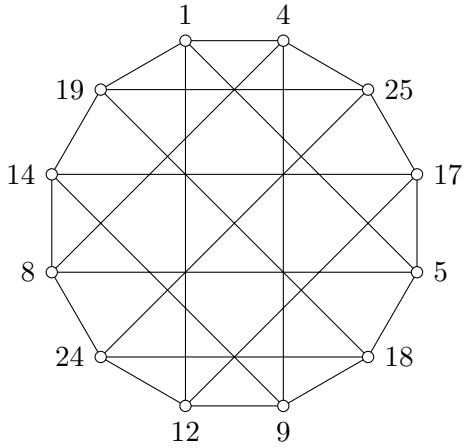


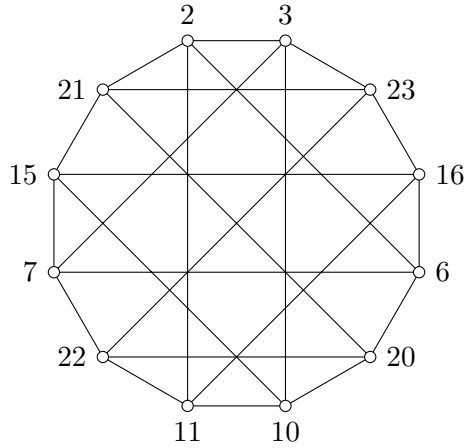
(a) 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 24 vrcholech varianta 1





(b) 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 24 vrcholech varianta 2

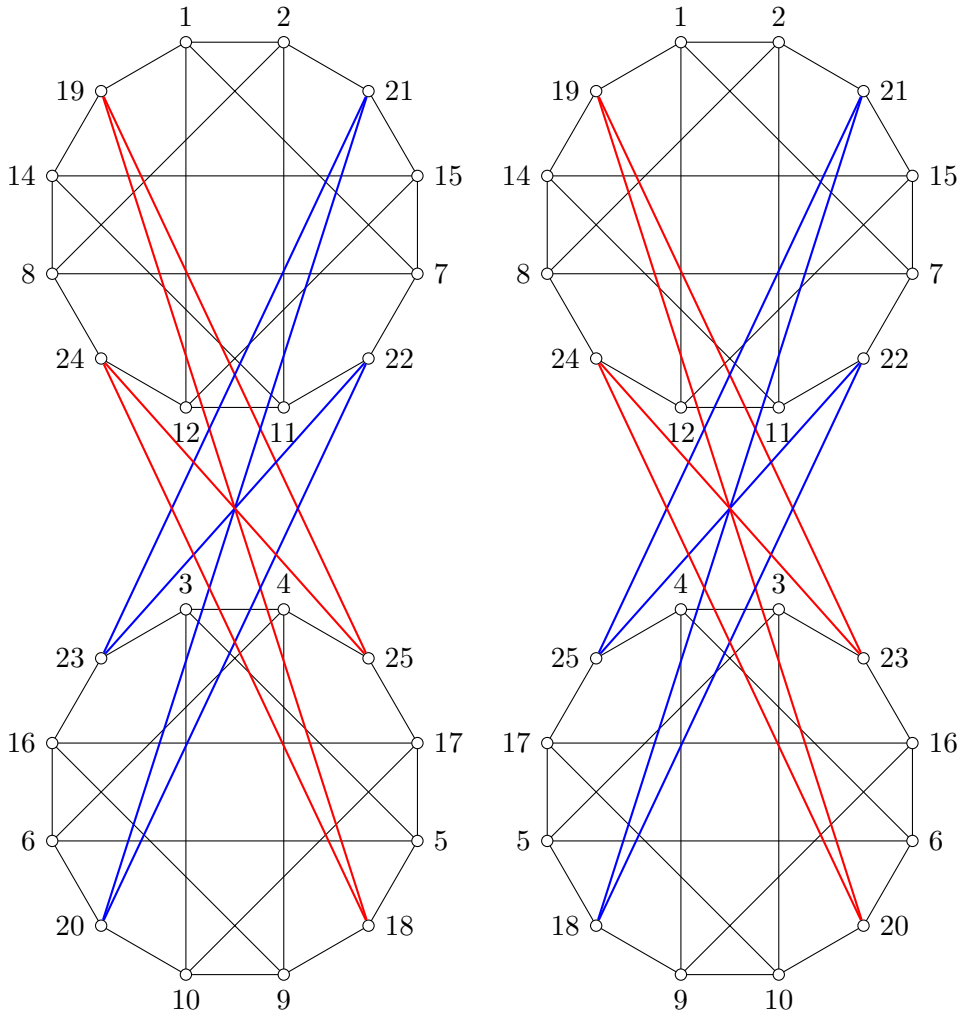




(c) 4-pravidelný graf s rozšířeným hendikepovým ohodnocením na 24 vrcholech varianta 3

Obrázek 33: Tři různá rozšířená hendikepová ohodnocení 4-pravidelných grafů na 24 vrcholech při ohodnocení f

Další věcí, které jsme si všimli během zkoumání 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením mající dvě a více komponent grafu H, je možnost různě propojovat čtyři a více komponent C_6 do jediné komponenty. Uvedeme dva příklady propojení dvou komponent H z grafu který je uveden v Obrázku 33 (a), kde např. odebereme cykly C_4 z komponent H vedoucí přes labely $\{19, 21, 24, 22\}$ a $\{23, 25, 20, 18\}$. Přidáme jiné C_4 cykly, tak aby váhy v těchto cyklech si navzájem odpovídaly se stejnou váhou vzhledem k faktoru H''. Kde např. label s hodnotou 19 měl sousední labely 21 a 22, což dává v součtu číslo 43, ale také nově přidaní sousedé k labelu 19, kteří jsou 25 a 18 dávají stejný součet 43, což můžeme vidět v Obrázku 34 (a).



(a) Variant I. (b) Variant II.

Obrázek 34: Dva rozšířené hendikepové 4-pravidelné grafy na 24 vrcholech varianta 4

Je snadné ověřit, že váha každého vrcholu i je $(40 + i)$.

5 Program pro hledání 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením

Cílem počítačového programu je nalézt pro zadaný počet vrcholů všechny unikátní 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením. Výsledky tohoto programu jsme využili ke zkoumání existence 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na daném počtu vrcholů. Dosažené výsledky tohoto programu jsou shrnuty a zmíněny v kapitole Dosažené výsledky 5.6.

Abychom mohli v tomto textu posuzovat rychlost algoritmů mezi sebou, připomeňme pojem asymptotické složitosti.

Definice 17 *[17] Nechť $f, g : \mathbb{N} \to \mathbb{R}$. Řekněme, že funkce $f(n)$ je třídy $\mathcal{O}(g(n))$, jestliže*

$$(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) : f(n) \leq c \cdot g(n).$$

Funkci $g(n)$ se pak říká asymptotický horní odhad funkce $f(n)$.

Poznámka 9 Nadále funkci $\log(n)$ v asymptotické složitosti $\mathcal{O}(\log(n))$ rozumíme takovou funkci $\log(n)$ o základu dva. Dodefinujme okrajovou hodnotu, pokud je $n = 0$, tak je i výraz $\log(0)$ nulový, kde $n \in \mathbb{N} \cup \{0\}$.

5.1 Volba programovacího jazyka

Pro realizaci programu jsme se rozhodli použít jazyk C++ ve verzi C++11. Důvodů proč jsme se rozhodli vybrat právě tento jazyk bylo hned několik. Klíčovou vlastností jazyka je podpora objektově orientovaného paradigmatu. Toto paradigma je vhodné, pro rozsáhlé projekty, kde je potřeba spravovat větší množství kódu, snadno a rychle rozšiřovat vnitřní logiku programu o nové funkce zvané metody. Dále pak možnost logicky spojovat části programu do takzvaných modulů, které se v tomto paradigmatu nazývají třídy. Mezi další výhodou jazyka C++ je podpora paralerizace, která umožňuje zrychlit vykonávání náročných části programu. Výhodou je i snadná přenositelnost kódu mezi různými operačními systémy. Posledním důvodem byla míra zkušeností s tímto jazykem, které jsem získal během studia.

Po zmíněných výhodách přichází čas uvést nevýhody. Nevýhodou jazyka C++11 oproti jiným (např. Java, Python) může býti vynaložení většího množství kódu při psaní stejného programu. S tím je i spojené větší riziko vzniku chyb. Tento fakt záleží i na zkušenosti programátora, který daný kód píše. Nevýhodou je i nutná správa paměti, která je ponechána na programátorovi, aby problémy vzniklé s využitím dynamické paměti vyřešil. Výhodou verze C++11 je přidání podpory inteligentních ukazatelů takzvaných smart pointers, které částečně řeší správu paměti

tím, že při zániku objektu na který odkazují, uvolní místo automaticky, bez nutnosti zásahu programátora. Vyhneme se tak jedné z nejčastějších chyb s neuvolněnou paměti zvanou memmory leak. Druhou výhodou použití inteligentních ukazatelů je částečné hlídaní přístupu k dynamicky vytvořeným proměnným. Toto hlídání zabraňuje přístupu k neexistujícím objektům a tím vzniku chyby zvané segmentation fault za podmínky, že uživatel pracuje s ukazateli podle doporučených rad.

```
// Smart pointers
#include <memory>
#include "ExtendedHandicapGraph"
#include "NSumProblemSolver"

int main()
{
   const int VERTICES = 12;
   const int REGULAR_GRAPH = 4;
   auto graph = std::make_shared(
       new ExtendedHandicapGraph(VERTICES, REGULAR_GRAPH)
   );
   auto solver = std::unique_ptr<NSumProblemSolver<int>>(
       new NSumProblemSolver<int>()
   );

   return 0;
}
```

Výpis 1: Použití chytrých ukazatelů v C++11

Pro sestavovací nástroj jsme se rozhodli použít program CMake ve verzi CMake 3.0.7. Jedná se o nástroj, který v základním nastavení nabízí pohodlné vygenerování Makefile souboru s nadefinovanými parametry při kompilaci programu. Konfiguračním souborem je soubor CMakeList.txt. Tento nástroj jsme využívali díky podpoře vývojového prostředí CLion, ve kterém program vznikal.

5.2 Vstup a výstup programu

Vstupem programu jsme zvolili parametr počet vrcholů 4-pravidelného grafu. Jedná se o přirozené číslo, které uživatel zadá pomocí klávesnice na vstup programu. Program ověří validnost vstupu. Pokud vstup byl validní program vypíše základní informace o zadaném grafu.

1. Počet vrcholů grafu

2. Počet hran v grafu
3. Vypočtená konstanta ℓ
4. Chybějící label v ohodnocení

Pak začne samotné hledání validního ohodnocení. Pokud program nalezne řešení pro zadaný počet vrcholů v grafu, vypíše se na nový řádek za znakem # číslo nalezeného řešení. Následně pak se na dalších n řádcích kde n je počet vrcholů, vypíše na každý řádek váha a k ní příslušné čtyři sousední váhy. Konec výpočtu programu je signalizován vytisknutým řetězcem **END**.

5.3 Popis algoritmů a datových struktur

V současné době není znám deterministický algoritmus na hledání r-pravidelných grafů s rozšířených hendikepových ohodnocením. Podařilo se nám v této práci navrhnout algoritmus, který hledá všechny 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením.

Předpokládejme, že známe celou množinu labelů pro 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením na n vrcholech. Také umíme vypočítat konstantu ℓ 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením za pomocí vzorce uvedeného ve Větě 4. Z Definice 16 je patrné, že pro každý vrchol 4-pravidelného grafu, který má label l, je dána podmínka, o součtu čtyř sousedních labelů, který musí býti rovny číslu $\ell+l$. Tyto čtyři sousední labely náleží čtyřem sousedním vrcholům s vrcholem majícím label l.

Algoritmus který uvedeme, pracuje pouze s množinou labelů rozšířeného hendikepového ohodnocení. Každému labelu l z této množiny zkoušíme přiřadit možné čtyři sousední labely pro 4-pravidelný graf tak, aby v součtu tyto sousední labely dávaly číslo $\ell + l$. Pokud nelze přiřadit labelu l sousední labely tak, aby součet čtyř sousedních labelů se rovnal číslu $\ell + l$, vrátí se algoritmus na poslední ohodnocený label a zkusí tomuto labelu přiřadit jinou validní kombinaci sousedních labelů. V nalezeném řešení musí každý label l mít přiřazené právě čtyři sousední labely, které v součtu musí dát číslo $\ell+l$. Tato metoda řešení úlohy se nazývá backtracking. Českým ekvivalentním výrazem bychom metodu backtracking nazvali jako metoda omylů a oprav. Výhodou použití metody backtracking je to, že odpadá nutnost zkoušet všechny řešení oproti metody hrubé síly zvané bruteforce, protože sestavujeme a testujeme pouze přípustné řešení. Důležité je si uvědomit, že nalezený ohodnocený 4-pravidelný graf, který je reprezentován pomocí seznamů čtyř sousedních labelů je isomorfní s nějakým 4-pravidelným grafem na n vrcholech, kde mohutnost množiny labelů je n. Podobný přístup by šel použit i na jiné úlohy, pro které bychom hledali všechny hendikepové nebo rozšířené hendikepové ohodnocení r-pravidelných grafů na n vrcholech.

Poznámka 10 Pro snadné pochopení předpokládejme globální přístup k proměnným vytvořeným v inicializaci Algoritmu 1. Je tedy možné k těmto proměnným přistupovat ve všech pro-

cedurách. Pro samotnou implementaci je vhodnější se samotným globálním proměnným zcela vyhnout. Zamezíme tím nechtěnému přepisu dat z více míst programu. Možnost, jak elegantně nahradit globální proměnné, nabízí objektové programování a to pomocí zapouzdření proměnných a vytvoření přístupových metod k těmto objektům. Komunikace mezi objekty je pak realizovaná pomocí těchto přístupových metod. Tímto způsobem budeme mít modifikaci dat pod kontrolou. Bude pak existovat jen jedno místo, přes které bude možné změnit data v objektu i v programu.

Poznámka 11 K popisu algoritmu jsme se rozhodl použít pseudokód. Důvod je ten, že samotný algoritmus není závislý na programovacím jazyku. Při pojmenování proměnných a procedur jsme se drželi anglického pojmenování a konvence camelCase. Rovněž číslování prvků v poli začíná od jedničky kvůli snadnému pochopení.

5.3.1 Inicializace algoritmu

Algoritmus 1 Inicializace algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

1: solution $\leftarrow$ Matrix type of $\mathbb{N}^{4\times n}$
2: filledLabelsInSolution $\leftarrow 0$
3: countOfUniqueSolutions $\leftarrow 0$
4: **for all** label $l \in$ labels **do**
5: missingNeighboursLabelsSum[l] $\leftarrow \ell + l$
6: countOfMissingNeighbours[l]$\leftarrow 4$
7: **end for**

Rozebereme si podrobně Algoritmus 1, který se týká inicializace. Na 1. řádku máme proměnnou *solution*, která nám bude představovat řešení rozšířeného hendikepového ohodnocení 4-pravidelného grafu. Jedná se o matici přirozených čísel o rozměrech $4 \times n$. Rozměr 4 jsme určili z předpokladů 4-pravidelného grafu, což značí informaci o čtyřech sousedních vrcholech každého vrcholu, ale i o čtyřech sousedních labelech každého labelu. Z předpokladů také známe informaci o počtu vrcholů, kterých je n. Počet labelů je také n. Proto je druhý parametr matice n. Na 2. řádku se nachází proměnná *filledLabelsInSolution*, která v sobě uchovává informaci o aktuálním počtu vyplněných sousedních labelů v matici *solution*. Pokud proměnná *filledLabelsInSolution* nabude hodnoty $4n$, budeme vědět, že všechny labely v matici *solution* mají vyplněny všechny své čtyři sousední labely a je možné rozhodnout, zda jsme našli další rozšířené hendikepové ohodnocení. Na 3. řádku se nachází proměnná *countOfUniqueSolutions*, která v sobě uchovává počet nalezených řešení. Na 4. až 6. řádku máme cyklus, který prochází přes všechny prvky množiny labelů a nastavuje pole *missingNeighboursLabelsSum* a *countOfMissingNeighbours*. Pole *missingNeighboursLabelsSum* v sobě uchovává aktuálně chybějící součet nepřiřazených labelů v matici *solution* pro daný label. Druhé pole *countOfMissingNeighbours* uchovává aktuální počet nepři-

řazených sousedních labelů pro každý label v matici *solution*. Tímto bychom měli inicializaci algoritmu probranou.

5.3.2 Rekurzivní procedura Labeling

Algoritmus 2 Rekurzivní procedura Labeling algoritmu pro hledání všech rozšířených hendikepových ohodnocení 4-pravidelných grafů na n vrcholech

```
1: procedure LABELING(currentLabel)
2:     if currentLabel ∉ labels then
3:         return Nill
4:     end if
5:     missNeighbours ← countOfMissingNeighbours[currentLabel]
6:     missingSum ← missingNeighboursLabelsSum[currentLabel]
7:     for all label l ∈ labels do
8:         if countOfMissingNeighbours[l] ≠ 0 ∧ l ≠ currentLabel then
9:             canUseLabel[l]← true
10:        else
11:            canUseLabel[l]← false
12:        end if
13:    end for
14:    combinations ← ComputeCombinations(canUseLabel, missNeighbours, missingSum)
15:    for all labeling-combination combination ∈ combinations do
16:        for all label neighbourLabels ∈ combination do
17:            MarkLabelsToSolution(currentLabel, neighbourLabel)        ▷ Call procedure
18:        end for
19:        if filledLabelsInSolution = 4n then
20:            if CheckSolution() then                                   ▷ Call procedure
21:                countOfUniqueSolutions ← countOfUniqueSolutions + 1
22:                PrintSolution()                                       ▷ Call procedure
23:            end if
24:        else
25:            nextLabel ← SelectNextLabel()
26:            Labeling(nextLabel)                                       ▷ Call recursion
27:        end if
28:        for all label neighbourLabel ∈ combination do
29:            UnmarkLabelsFromSolution(currentLabel, neighbourLabel)    ▷ Call procedure
30:        end for
31:    end for
32: end procedure
```

Nyní popíšeme jádro celého algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením a to rekurzivní proceduru *Labeling* v Algoritmu 2. Jejím úkolem je hledat nové řešení. Vstupem procedury je label označen *currentLabel*. Cílem této funkce je přiřadit tomuto labelu *currentLabel* jeho zbylé sousední labely tak, aby součet všech jeho čtyř sousedních labelů v příslušném sloupci matice *solutuion* dával v součtu číslo (ℓ+*currentLabel*).

Pak na základě podmínky o vyplněném počtu labelů v proměnné *solution* se rozhodne, zda je potřeba přiřadit další sousední labely chybějícím labelům v matici *solution*, nebo zkontrolovat a vytisknout řešení (viz 19. až 27. řádek).

Na 2. řádku se provede test validity labelu *currentLabel*. Pokud test určí, že label není validní, ukončí se běh procedury *Labeling*. Tato nevalidní situace může vzniknout, pokud by se při prvním volání této procedury vložil nevalidní label. V 5. řádku se přiřadí do proměnné *missNeighbours* počet nepřiřazených sousedních labelů vzhledem k labelu *currentLabel*, které je potřeba určit, aby label *currentLabel* měl přiřazené právě čtyři sousední labely. Na 6. řádku se provede přiřazení chybějícího součtu nepřiřazených labelů sousedů vzhledem k proměnné *currentLabel*. Pak následuje cyklus na 7. až 13. řádku, který prochází přes množiny všech labelů. V tomto cyklu se testuje, zda konkretní label l, může býti sousedním labelem k labelu *currentLabel*. Příslušná informace se zapíše do pole *canUseLabel*. Na 14. řádku se zavolá procedura *ComputeCmbinations*, která ze zadaných parametrů *canUseWeight*, *missNeighbours* a *missingSum*, vypočte počet možných kombinací a sestaví přípustné kombinace m-tic labelů, kde m je číslo *missNeighbours*. Číslo m může nabývat pouze jedné z hodnot $\{1, 2, 3, 4\}$ pro 4-pravidelné grafy. Každá přípustná kombinace musí splňovat takové podmínky, že součet všech labelů v dané kombinaci musí dát číslo *missingSum* a navíc všechny labely musí mít nastavenou hodnotu *true* v poli *canUseLabel*. Příslušné kombinace se uloží do pole *combinations*. Nyní mohou nastat pouze dvě situace.

První situace je taková, že existuje alespoň jedna kombinace a je tedy možné přiřadit sousední labely k labelu *currentLabel*. Nebo nastane druhá situace, při které neexistuje žádná přípustná kombinace. V takovém případě procedura *Labeling* skončí. Cyklus, který se nachází na 15. až 30. řádku, má na starosti postupné procházení přípustných kombinací sousedních labelů, uložených v proměnné *combinations*. Každá kombinace m-tic labelů je přiřazena k aktuálnímu labelu *currentLabel*. To můžeme vidět v cyklu, který se nachází na 16. až 18. řádku. O samotné přiřazení nových sousedních labelů z proměnné *combination* k labelu *currentLabel* a naopak se stará procedura *MarkLabelsToSolution*. Vstupními argumenty této procedury je dvojice labelu *currentLabel* a *neighbourLabel*, kterým se doplní sousední label. Volání této procedury vidíme na 17. řádku. Na 19. řádku proběhne kontrola o zaplněnosti matice *solution*. Pokud je matici *solution* plně zaplněná a tedy každý label má přiřazené právě čtyři sousední labely, pak se zavolá procedura *CheckSolution*, která zkontroluje chybějící součty v poli *missingNeighboursLabelsSum*, pro každý label, zda je hodnota nulová. V případě úspěšného vyhodnocení procedury *CheckSolution*, zvýší se počet nalezených rozšířených hendikepových ohodnocení o jedničku v proměnné *countOfUniqueSolutions* na řádku 19. Na 20. řádku se pak procedura *PrintSolution* postará o vytisknutí množiny labelů spolu s odpovídající množinou sousedních labelů ke každému labelu. Pokud zatím nemají všechny labely přiřazené právě čtyři sousední labely v matici *solution*. Procedura *SelectNextLabel* vybere label, kterému chybí sousední label a uloží ho do proměnné *nextLabel* na řádku 25. Pak se zavolá procedura *Labeling* na 26. řádku s

parametrem *nextLabel* a začne doplňování sousedních labelů k labelu *nextLabel*. Zde se nachází rekurzivní volání. Poslední cyklus se v této proceduře nachází na 28. až 30. řádku. Jeho úkolem je projít všechny sousední labely, které jsou uložené v proměnné *combination* a odebrat tak tuto kombinaci sousedních labelů vzhledem k labelu *currentLabel* z aktuálních struktur proměnných *solutions*, *countOfMissingNeighbours*, *missingNeighboursLabelsSum* a upravit počet vyplněných labelů v proměnné *filledLabelsInSolution*. O tento krok se stará procedura *UnmarkLabelsFrom-Solution*, jejími parametry jsou proměnné *currentLabel* a *neighbourLabel*. Důvod proč odebíráme aktuální kombinaci je ten, aby bylo možné vyzkoušet novou kombinaci a nalézt tak i jiné řešení. Tímto bychom měli, hlavní jádro algoritmu probrané.

5.3.3 Procedura MarkLabelsToSolution

Popišme proceduru *MarkLabelsToSolution*, na kterou jsme se odkazovali v Algoritmu 2. Jak již bylo zmíněno, tato procedura se stará o propojení dvou labelů, které se mají stát sousedními labely mezi sebou. To znamená, že pro tyto dva labely je potřeba upravit matici *solution* sousedních labelů, taktéž snížit počet nepřiřazených sousedních labelů v poli *countOfMissing-Neighbours* o jedničku, zbývá také upravit součet zbývajících labelů v poli *missingNeighboursLa-belSum* a v neposlední řádě zvýšit počet vyplněných labelů v proměnné *filledLabelsInSolution* o dva. Dále víme, že relace býti sousedním labelem k labelu je symetrická relace. Proto v matici *solution* přiřadíme labelu *currentLabel* sousední label *neighbourLabel* a labelu *neighbourLabel* přiřadíme sousední label *currentLabel*.

Algoritmus 3 Procedura MarkLabelsToSolution algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

```
1:  procedure MarkLabelsToSolution(currentLabel, neighbourLabel)
2:      missingNeighbours1 ← countOfMissingNeighbours[currentLabel]
3:      missingNeighbours2 ← countOfMissingNeighbours[neighbourLabel]
4:      solution[missingNeighbours1][Index(currentLabel)] ← neighbourLabel
5:      solution[missingNeighbours2][Index(neighbourLabel)] ← currentLabel
6:      filledLabelsInSolution ← (filledLabelsInSolution + 2)
7:      countOfMissingNeighbours[currentLabel] ← (missingNeighbours1 − 1)
8:      countOfMissingNeighbours[neighbourLabel] ← (missingNeighbours2 − 1)
9:      sum1 ← missingNeighboursLabelsSum[currentLabel]
10:     sum2 ← missingNeighboursLabelsSum[neighbourLabel]
11:     missingNeighboursLabelsSum[currentLabel] ← (sum1 − neighbourLabel)
12:     missingNeighboursLabelsSum[neighbourLabel] ← (sum1 − currentLabe)
13: end procedure
```

Na 2. a 3. řádku si do příslušných proměnných *missingNeighbours1* a *missingNeighbours2* uložíme počet nepřiřazených sousedů labelů *currentLabel* a *neighbourLabel*. Ve 4. řádku přiřadíme labelu *currentLabel* sousední label *neighbourLabel* a na 5. řádku přiřadíme labelu *neighbourLabel* sousední label *currentLabel* v matici *solution*. Protože 2. rozměr matice *solution* je n a ne všechny

labely odpovídají prvkům v množině indexů matice *solution*, která je rovna $\{1, 2, \ldots, n-1, n\}$. Zavedeme proto proceduru *Index*, která nám vytvoří injektivní zobrazení z množiny labelů do množiny indexů matice *solution*. Pomocí procedury *Index* budeme schopni přistupovat ke sloupci daného labelu v matici *solution*. Na 6. řádku přičteme dvojku k počtu vyplněných labelů v proměnné *filledLabelsInSolution*. Na 7. a 8. řádku upravíme počty zbylých nepřiřazených sousedních labelů pro dané labely *currentLabel* a *neighbourLabel*. Na 9. a 10. řádku si do pomocných proměnných *sum1* a *sum2* uložíme neaktuální součet chybějících sousedních labelů, který vzápětí na 11. a 12. řádku upravíme podle aktuálních dat v matici *solution* pro každý label *currentLabel* a *neighbourLabel*. Tímto bychom měli proceduru *MarkLabelsToSolution* probranou.

5.3.4 Procedura UnmarkLabelsFromSolution

Popišme proceduru *UnmarkLabelsFromSolution* na kterou jsme se také odkazovali v Algoritmu 2. Úkol této procedury je přesně opačný od procedury *MarkLabelsToSolution* a tedy odebrat stávající dvojici sousedních labelů *currentLabel* a *neighbourLabel* z matice *solution*. Dale pak zvětšit počet nepřiřazených sousedních labelů v poli *countOfMissingNeighbours* o jedničku, následně upravit chybějících součet nepřiřazených labelů v proměnné *missingNeighboursLabelsSum* a snížit počet přiřazených labelů o dva v proměnné *filledLabelsInSolution*.

Algoritmus 4 Procedura UnmarkLabelsFromSolution algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

```
1:  procedure UnmarkLabelsFromSolution(currentLabel, neighbourLabel)
2:      missingNeighbours1 ← (countOfMissingNeighbours[currentLabel] + 1)
3:      missingNeighbours2 ← (countOfMissingNeighbours[neighbourLabel] + 1)
4:      solution[missingNeighbours1][Index(currentLabel)] ← 0
5:      solution[missingNeighbours2][Index(neighbourLabel)] ← 0
6:      filledLabelsInSolution ← (filledLabelsInSolution − 2)
7:      countOfMissingNeighbours[currentLabel] ← missingNeighbours1
8:      countOfMissingNeighbours[neighbourLabel] ← missingNeighbours2
9:      sum1 ← missingNeighboursLabelsSum[currentLabel]
10:     sum2 ← missingNeighboursLabelsSum[neighbourLabel]
11:     missingNeighboursLabelsSum[currentLabel] ← (sum1 + neighbourLabel)
12:     missingNeighboursLabelsSum[neighbourLabel] ← (sum1 + currentLabel)
13: end procedure
```

Na 2. a 3. řádku si do proměnných *missingNeighbours1* a *missingNeigbours2* uložíme počet zbývajících nepřiřazených labelů *currentLabel* a *neighbourLabel* zvýšený o jedničku. Tyto nové proměnné nám uvádí i informaci o posledním vyplněném řádku pro daný label v matici *solution*, který chceme odebrat. Na 4. a 5. řádku Algoritmu 4 přiřadíme hodnotu 0 posledně vloženým sousedním labelům *currentLabel* a *neighbourLabel*. Na 6. řádku snížíme počet vyplněných labelů o dva v proměnné *filledLabelsInSolution*. Také zvýšíme počet zbývajících sousedních labelů v poli *countOfMissingNeighbours* o jedničku na 7. řádku pro label *currentLabel* a na 8. řádku pro

label *neighbourLabel*. Na 9. a 10. řádku si uložíme součet nepřiřazených labelů, který upravíme na 11. a 12. řádku pro labely *currentLabel* a *neighbourLabel*, aby informace v poli *missingNeighboursLabelsSum* byly korektní a shodovaly se s daty v matici *solution*. Tímto bychom měli proceduru *UnmarkLabelsFromSolution* probranou.

5.3.5 Procedura SelectNextLabel

Popišme proceduru *SelectNextLabel*, na kterou jsme se odkazovali v Algoritmu 2. Jejím úkolem je vybrat label, který ještě nemá přiřazené všechny čtyři sousední labely. Navíc budeme požadovat od této procedury, aby vybrala takový label, který má nejmenší nenulovou kladnou hodnotu v poli *countOfMissingNeighbours*. Důvodem tohoto požadavku je jistá budoucí náročnost výpočtu m-tic v proceduře *ComputeCombination*, kde výpočet m-tice je snazší, když parametr m je co nejmenší. V Algoritmu 2 má jistě smysl volat tuto proceduru, protože proměnná *filledLabelsInSolution* nemá hodnotu $4n$, což je hodnota plně zaplněné matice *solution* labely. Pak určitě existují minimálně dva takové labely, kterým chybí alespoň jeden sousední label v matici *solution*.

Algoritmus 5 Procedura SelectNextLabel algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

```
 1: procedure SelectNextLabel
 2:     nextLabel ← −1
 3:     minMissingNeighbourCount ← 5;
 4:     for all label l ∈ labels do
 5:         missingCount ← countOfMissingNeighbours[l]
 6:         if missingCount ≠ 0 ∧ missingCount < minMissingNeighbourCount then
 7:             nextLabel ← l
 8:             minMissingNeigbourCount ← missingCount
 9:         end if
10:     end for
11:     return nextLabel
12: end procedure
```

Na 2. řádku Algoritmu 5 si do pomocné proměnné *nextLabel* přiřadíme výchozí hodnotu -1. V této proměnné budeme ukládat labely, které nemají přiřazené všechny čtyři sousední labely v matici *solution*. Na 3. řádku si do proměnné *minMissingNeighbourCount* uložíme výchozí hodnotu 5. Tato proměnná bude v sobě uchovávat nejmenší nalezený počet nepřiřazených sousedů labelu *nextLabel*. Na 4. až 9. řádku hledáme nejmenší nenulovou a kladnou hodnotu prvku v poli *countOfMissingNeighbours*. Po dokončení hledání nejmenšího prvku v poli *countOfMissingNeighbours* máme v proměnné *nextLabel* uloženého kandidáta na další ohodnocení v proceduře *Labeling*. Proto ho na 11. řádku vrátíme jako výsledek této procedury.

5.3.6 Procedura CheckSolution

Existuje mnoho způsobů jak ověřit správnost řešení. Zvolili jsme implementačně nejjednodušší řešení a to pouze ověřit, že hodnota v poli *missingNeighboursLabelsSum* je pro každý label nulová. Protože víme, že v matici *solution* jsou vyplněny všechny labely což je jedním z předpokladů pro řešení, má tedy v této chvíli smysl zkontrolovat pole *missingNeighboursLabelsSum* zda je zbývající součet sousedních labelů nulový pro každý label.

Algoritmus 6 Procedura CheckSolution algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

```
1: procedure CheckSolution
2:     isSolutionCorrect ← true
3:     for all label l ∈ labels do
4:         if isSolutionCorrect ∧ missingNeighboursLabelsSum[l] != 0 then
5:             isSolutionCorrect ← false
6:         end if
7:     end for
8:     return isSolutionCorrect
9: end procedure
```

Výsledek o nulovosti pole *missingNeighboursLabelsSum* je uložen v proměnné *isSolutionCorrect*. Na 8. řádku pak vrátíme proměnou *isSolutionCorrect*, ve které je informace, zda se jedná o rozšířené hendikepové ohodnocení.

5.3.7 Procedura PrintSolution

Procedura *PrintSolution* se stará o vytisknutí řešení rozšířeného hendikepového ohodnocení 4-pravidelného grafu. Pro každý label l vytiskne jeho čtyři sousední labely, které jsou uloženy v pomocných proměnných n_1, n_2, n_3 a n_4.

Algoritmus 7 Procedura PrintSolution algoritmu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na n vrcholech

```
1: procedure PrintSolution
2:     print "# countOfUniqueSolutions"
3:     for all label l ∈ labels do
4:         n1 ← solution[1][Index(l)]
5:         n2 ← solution[2][Index(l)]
6:         n3 ← solution[3][Index(l)]
7:         n4 ← solution[4][Index(l)]
8:         print "l  n1 n2 n3 n4"
9:     end for
10: end procedure
```

5.3.8 Sestavení programu

Nyní máme skoro všechny potřebné algoritmy probrané, kromě procedury *ComputeCombination*, která si díky svému rozsahu zasloužila o samostatnou sekci, protože se jedná o rozsáhlý úkol řešení k-SUM úlohy, kde $1 \leq k \leq 4$. V Algoritmu 8 vidíme, že na 1. řádku načteme počet vrcholů grafu do proměnné n. Na 2. až 8. řádku je inicializace programu z Algoritmu 1. Na 8. řádku spustíme rekurzivní proceduru *Labeling* s parametrem 1, což značí že budeme ohodnocovat právě label s hodnotou 1. Po ukončení procedury *Labeling* v Algoritmu 8 vytiskneme výraz **END**, čímž signalizujeme uživateli konec běhu programu. V programu ovšem nezapomeneme ošetřit krajní možnosti vstupních dat.

Algoritmus 8 Sestavení programu pro hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením

```
1: Read(n)                                          ▷ Read number of vertices
2: solution ← Matrix type of ℕ^(4×n)
3: filledLabelsInSolution ← 0
4: countOfUniqueSolutions ← 0
5: for all label l ∈ labels do
6:     missingNeighboursLabelsSum[l] ← (ℓ + l)
7:     countOfMissingNeighbours[l]← 4
8: end for
9: Labeling(1)
10: Print("END")
```

5.4 k-SUM úloha

V této části se budeme zabývat takzvanou k-SUM úlohou. Důvod proč se zabýváme k-SUM úlohou je takový, že právě metoda *ComputeCombinations* zmíněná v Algoritmu 2, je implementací k-SUM úlohy. Dříve než uvedeme definici k-SUM úlohy, uveďme definici subset sum úlohy, ze které úloha k-SUM vychází.

Definice 18 *[8]* ***Subset sum úloha*** *Mějme uspořádanou dvojici (S,t), kde $S = \{x_1, x_2, \ldots, x_n\}$ je konečná a neprázdná množina, prvky $x_i \in \mathbb{N} \wedge 1 \leq i \leq n$ a číslo $t \in \mathbb{N}$. Ptáme se, zda existuje podmnožina $S' \subseteq S$ taková, že součet všech prvků v S' dává číslo t.*

Poznámka 12 Z teorie složitosti je známo, že úloha subset sum patří do speciální třídy NP úloh. Pro úlohy z NP třídy je v současně době znám pouze exponenciální algoritmus pro nalezení řešení. Pro ověření výsledku úlohy, zda je řešením, existuje polynomiální algoritmus.

Definice 19 ***k-SUM úloha*** *Mějme uspořádanou trojici (S,k,t), takovou kde $S = \{x_1, x_2, \ldots, x_n\}$ je konečná a neprázdná množina, prvky $x_i \in \mathbb{N} \wedge 1 \leq i \leq n$ a čísla $t, k \in \mathbb{N}$. Ptáme se, zda existuje podmnožina $S' \subseteq S$ o mohutnosti k taková, že součet všech k prvků v S' dává číslo t.*

Poznámka 13 V některých článcích např [9] nebo [10] se čtenář může dočíst, že autoři k-SUM úlohu chápou trochu jinak, než jsme zavedli v tom to textu. Mají předpoklad, že prvky množiny S^*, mohou být celá čísla a ptají se, zda existuje podmnožina $S^{*\prime} \subseteq S^*$ o mohutnosti k kde součet k čísel z množiny $S^{*\prime}$ je nulový. Pokud by chtěl čtenář využít alternativní formulaci, stačí mu vytvořit novou množinu S^* tímto způsobem: $S^* = \{x_i - t, x_2 - t, \ldots, x_n - t\}$, kde prvky $x_i \in S$ a t je hledaný součet čísel. Pak se může ptát, jestli množina S^* obsahuje podmnožinu $S^{*\prime} \subseteq S^*$ o mohutnosti k, kde součet k čísel z množiny $S^{*\prime}$ je nulový.

Poznámka 14 Praktické využití například 3-SUM problémů můžeme nalézt ve výpočetní geometrii v článku [10] se autoři zmiňují např. o úloze GeomBase. V této úloze se autoři ptají, zda existuje v dané množině trojice celočíselných kolineárních bodů v rovině.

V našem textu nás budou zajímat pouze úlohy 1-SUM, 2-SUM , 3-SUM a 4-SUM. Prvky množiny S budou tvořit labely rozšířeného hendikepového ohodnocení. V této množině budeme hledat všechny podmnožiny S' k-tic labelů, které v součtu dají číslo t. Tento požadavek je přísnější než definice k-SUM úlohy, kde cílem bylo rozhodnout, zda existuje podmnožina $S' \subseteq S$ obsahující různé k čísla, které v součtu dávají číslo t. Zkusíme se podívat na různé algoritmy a pokusit se při nalezení řešení neukončit běh algoritmu, ale pokračovat v dalším hledání, dokud nenajdeme všechny k-tice labelů, které budou v součtu dávat číslo t.

5.4.1 1-SUM úloha

Není těžké si uvědomit, že u 1-SUM úlohy stačí rozhodnout, zda label s hodnotou t je prvkem množiny S. Proto je hledaná pouze jednoprvková podmnožina S'. Algoritmicky existuje mnoho

různých řešení jak otestovat zda prvek t je prvek množiny S. Uvedeme zde pár možných řešení a k nim asymptotickou složitost.

1. Reprezentovat množinu S pomocí datové struktury pole. Testovat průchod polem, zda hodnota labelu v poli je rovna hodnotě t. Asymptotická složitost testování je $\mathcal{O}(n)$.
2. Reprezentovat množinu S pomocí datové struktury pole, ve které jsou hodnoty labelů uspořádány vzestupně. Využit algoritmu binárního půlení na pole a hledat label s hodnotou t. Asymptotická složitost testování je $\mathcal{O}(\log(n))$.
3. Reprezentovat množinu S pomocí datové struktury hash tabulky. Nalezení konkretního labelu s danou hodnotou má asymptotickou složitost $\mathcal{O}(\log(n))$.

Při použití těchto způsobů je i nutné dbát na složitost vytvoření dané datové struktury.

5.4.2 2-SUM úloha

2-SUM úloha bývá často pokládána na přijímacích pohovorech do různých firem zabývající se tvorbou softwaru. Proto 2-SUM úlohu můžeme nalézt například v předmětu Stanfordské univerzity CS9 Problem-Solving for the CS Technical Interview, kde se v dokumentu [11] nachází podrobné řešení různých metod řešení 2-SUM úlohy. Zde autoři ověřují existenci jedné dvouprvkové podmnožiny splňující součet t (v článku označena proměna k). Později metody uvedené v článku [11] rozebereme pečlivěji. Stejný typ úloh můžeme nalézt na webové stránce leetcode.com, která se zabývá úlohami kladenými při výběrových řízení pro programátory. I zde je rozebrána úloha 2-SUM. Důvodem proč uvádíme stránku leetcode.com je ten, že autora blogu [12] inspirovaly úlohy 2-SUM, 3-SUM a 4-SUM ze stránky leetcode.com natolik, že si dal tu práci a podrobně rozebral metody řešení těchto úloh v několika článcích s jistými variantami. V článku [12] autor rozebírá silnější variantu 2-SUM úlohy, která se týká hledání všech dvouprvkových podmnožin S'. Jeho postup využijeme v proceduře *ComputeCombination* při hledání všech přípustných dvojic labelů. Ukážeme si možné algoritmy řešení úlohy 2-SUM, kde budeme hledat všechny přípustné dvojice, které budeme ukládat do vhodné datové struktury např. nafukovací pole. Nazveme si tuto strukturu *Combinations*. Prostorovou složitost struktury *Combinations*, nebudeme započítat do prostorové složitosti algoritmu. Dále předpokládejme, že asymptotickou složitost uložení dvojice labelů do struktury $\mathcal{O}(1)$.

1. **Řešení úlohy 2-SUM pomocí metody hrubé síly**
   Množinu S reprezentujeme pomocí datové struktury pole uloženou v proměnné S. V tomto poli S testujeme všechny dvojice. Jedná se o modifikovanou metodu z sekce Brute force z článku [11], kterou uvádíme v Algoritmu 9. Asymptotická složitost Algoritmu 9 je $\mathcal{O}(n^2)$ za předpokladu, že procedura *Lenght* má asymptotickou složitost $\mathcal{O}(1)$. Prostorová složitost Algoritmu 9 je $\mathcal{O}(1)$. Labely v poli S nemusí být uspořádány. Tato metoda se dá využít pro řešení všech k-SUM úloh. Její nevýhodou je však vzrůstající asymptotická složitost s rostoucím k která je $\mathcal{O}(n^k)$

Algoritmus 9 Hledání řešení úlohy 2-SUM pomocí hrubé síly

```
1: procedure BruteForce2-SUM([ ] S, t)
2:     Combinations c
3:     for i ← 1; i ≤ Lenght(S); i← (i+1) do
4:         for j ← (i+1); j ≤ Lenght(S); j←(j+1) do
5:             if S[i] + S[j] = t then
6:                 c.savePair(S[i], S[j])
7:             end if
8:         end for
9:     end for
10:    return c
11: end procedure
```

2. **Řešení úlohy 2-SUM pomocí metody hašování**

   Opět množinu S reprezentujeme pomocí datové struktury pole uloženou v proměnné S. Dále vytvoříme prázdnou hašovací tabulku h. Začneme procházet všechny labely uložené v poli S a testovat, zda label s hodnotou $t - S[\mathrm{i}]$ se nenachází v hašovací tabulce. Pokud se tento label nenachází v hašovací tabulce, uložíme label $S[\mathrm{i}]$ do hašovací tabulky. Pak začneme testovat další label v poli S. Pokud se label $t - S[\mathrm{i}]$ nachází v hašovací tabulce, máme novou dvojici, kterou vytiskneme, ale i tak label $S[\mathrm{i}]$ vložíme do hašovací tabulky. Jedná se o modifikovanou metodu z sekce Hashing v článku [11], kterou uvádíme v Algoritmu 10. Asymptotická složitost je stejná jako v článku [11] a tedy $\mathcal{O}(n)$, což je způsobeno tím, že do hašovací tabulky n krát vkládáme labely a n krát klademe dotaz zda label s hodnoutou $t - S[\mathrm{i}]$ se nachází v hašovací tabulce. Autoři také zmiňují prostorovou složitost $\mathcal{O}(n)$. My však díky povaze labelů rozšířeného hendikepového ohodnocení jsme schopni snížit tento prostor s využitím bitového pole na $n + 1$ bitů, k reprezentaci daného labelu. Největší hodnota jakou labelu je $n + 1$ a nejmenší hodnota je 1. Číslo n je počet všech labelů z rozšířeného hendikepového ohodnocení.

Algoritmus 10 Hledání řešení úlohy 2-SUM pomocí metody hašování

```
1: procedure Hashing2-SUM([ ] S, t)
2:     HashTable h
3:     Combinations c
4:     for i ← 1; i ≤ Lenght(S); i← (i+1) do
5:         if h.Contains(t − S[i]) then
6:             c.savePair(S[i], t − S[i])
7:         end if
8:         h.Insert(S[i])
9:     end for
10:    return c
11: end procedure
```

3. **Řešení úlohy 2-SUM pomocí metody binárního půlení**

   Množinu S reprezentujeme pomocí datové struktury pole v proměnné S. Pro použití algoritmu binárního půlení je potřebné, aby labely v poli S byly uspořádány vzestupně podle své hodnoty. Pak je možné v poli S hledat label $(t - S[\mathrm{i}])$ s asymptotickou složitostí hledání $\mathcal{O}(\log(n))$. Protože se na dotaz existence labelu $(t - S[\mathrm{i}])$ ptáme $(n-1)$ krát, výsledná asymptotická složitost Algoritmu 11 je $\mathcal{O}(n \log(n))$. Prostorová složitost algoritmu je $\mathcal{O}(1)$. Uvádíme Algoritmus 11, který je odvozený z algoritmu z článku [11] ze 3. sekce *Sorting and Binary Search*.

Algoritmus 11 Hledání řešení úlohy 2-SUM pomocí metody binárního půlení

```
1: procedure BinarySearch2-SUM([ ] S, t)
2:     Combinations c
3:     for i ← Lenght(S); i > 1; Lenght(S); i←(i−1) do
4:         lhs ← 1
5:         rhs ← (i − 1)
6:         while lhs < rhs do
7:             m ← (lhs + rhs) / 2
8:             if S[m] < t − S[i] then
9:                 lhs ← (m + 1)
10:            else
11:                if S[m] > (t − S[i]) then
12:                    rhs ← (m − 1)
13:                else
14:                    c.savePair(S[i], (t − S[i]))
15:                end if
16:            end if
17:        end while
18:    end for
19:    return c
20: end procedure
```

4. **Řešení úlohy 2-SUM pomocí metody dvou ukazatelů**

   Množinu S budeme reprezentovat pomocí datové struktury pole v proměnné S. V poli S je potřebné dodržet vzestupné uspořádání labelů, aby bylo možné použít metodu dvou ukazatelů. Poté si vytvoříme dva ukazatele indexů lhs a rhs v poli S. Ukazatel lhs bude ukazovat na první label v poli S a ukazatel rhs bude ukazovat na poslední prvek v poli S. Pak začneme porovnávat součet dvou labelů $S[\mathrm{lhs}]$ a $S[\mathrm{rhs}]$ s číslem t, podle porovnání se určí, zda levý index lhs zvýšíme, nebo pravý index rhs snížíme, tak abychom se přiblížil k součtu t. Asymptotická složitost algoritmu je $\mathcal{O}(n)$, protože každý prvek navštívíme právě jednou. Prostorová složitost algoritmu je $\mathcal{O}(1)$. Algoritmus 12 jsme převzali z článku [12].

Algoritmus 12 Hledání řešení úlohy 2-SUM pomocí metody dvou ukazatelů

```
1: procedure TwoPointers2-SUM([ ] S, t)
2:     Combinations c
3:     lhs ← 1
4:     rhs ← |S|
5:     while lhs < rhs do
6:         sum ← (S[lhs] + S[rhs])
7:         if sum < t then
8:             lhs ← (lhs +1)
9:         end if
10:        if sum > t then
11:            rhs ← (rhs −1)
12:        end if
13:        if sum = t then
14:            c.savePair(S[lhs], (t − S[rhs]))
15:            rhs ← (rhs −1)
16:            lhs ← (lhs +1)
17:        end if
18:    end while
19:    return c
20: end procedure
```

Uvedli jsme čtveřici algoritmů, které řeší úlohu 2-SUM, ve které hledáme všechny dvouprvkové podmnožiny S', kde součet prvků v každé podmnožině S' je číslo t. Nejvhodnějšími algoritmy pro řešení této úlohy, jsou algoritmy založené na metodě hašování a metodě dvou ukazatelů. Obě tyto metody mají asymptotickou složitost $\mathcal{O}(n)$, která je z uvedených algoritmů nejmenší. Metoda dvou ukazatelů je lepší v prostorové složitosti, která je $\mathcal{O}(1)$, na rozdíl od metody založené na hašování, kde prostorová složitost je $\mathcal{O}(n)$.

5.4.3 3-SUM úloha

V Poznámce 14 jsme uvedli zajímavost, že úloha 3-SUM nachází uplatnění i ve výpočetní geometrii. V roce 1999 publikoval autor prof. Jeff Erickson článek [13], ve kterém uvádí dolní asymptotickou složitost $\Omega(n^{\lceil k/2 \rceil})$ pro třídu úloh do které patří i problém k-SUM (jedná se pouze o existenci k-prvkové podmnožiny S'). Metoda v článku [13] používá speciální výpočetní model k-linear decision tree. V předmětu Algorithmic Lower Bounds: Fun with Hardness Proofs prof. Erik Demaine představuje přehled dokázaných algoritmů pro 3-SUM úlohu spolu s asymptotickými složitostmi [14], kde naivním způsobem se dají tříprvkové množiny S' nalézt pomocí metody hrubé síly s asymptotickou složitostí $\mathcal{O}(n^3)$. Dalšími dvěma způsoby, které mají asymptotickou složitost $\mathcal{O}(n^2)$, pro nalezení všech tříprvkových podmnožin S' splňující nulový součet, ale i pro nalezení jednoho řešení, jsou metody založené na hašování a metodě dvou ukazatelů. Obě zmíněné metody se dají pozměnit na nalezení všech tříprvkových podmnožin splňující součet t. Tyto metody následně popíšeme pro 3-SUM úlohu. Dále v přednášce [14] je zmíněná

nejmenší nalezená asymptotická složitost $\mathcal{O}(n^{1.5}\log(n))$ pro algoritmus uvedený v článku [15], který řeší úlohu 3-SUM pro existenci tříprvkové množiny S' s nulovým součtem prvků v množině S'. Nejlepší dosud nalezená asymptotická složitost algoritmu, který ověří existenci (nenajde všechny 3-prvkové množiny S') úlohy 3-SUM je $\mathcal{O}(n^{1.5})$, která je uvedena v článku [16].

1. **Řešení úlohy 3-SUM pomocí metody dvou ukazatelů**
   Množinu S budeme reprezentovat pomocí datové struktury pole v proměnné S. V poli S je potřeba dodržet vzestupné uspořádání labelů. Poté si vytvoříme dva ukazatele indexů lhs a rhs v poli S. Uvedeme zde Algoritmus 13, který hledá všechny tříprvkové množiny S'. Součet prvků v těchto množinách S' dává číslo t. Pricip činnosti Algoritmu 13 je obdobný jako u Algoritmu 12 s tím rozdílem, že u Algoritmů 13 je navíc cyklus *for*, ve kterém testujeme, zda existuje v množině S label který v součtu má hodnotou $t-S$[i]$-S$[j] pomocí metody dvou ukazatelů. Asymptotická složitost Algoritmu 13 je $\mathcal{O}(n^2)$ a to proto, že $(n-2)$ krát voláme metodu dvou ukazatelů, která má asymptotickou složitost $\mathcal{O}(n)$. Prostorová složitost Algoritmu 13 je $\mathcal{O}(1)$.

Algoritmus 13 Hledání řešení úlohy 3-SUM pomocí metody dvou ukazatelů

```
1: procedure TwoPointers3SUM([ ] S, t)
2:     Combinations c
3:     for i ← |S|; i ≥ 3;  i ← (i−1) do
4:         lhs ← 1
5:         rhs ←(i −1)
6:         lookingLabel← (t − S[i])
7:         while lhs < rhs do
8:             sum ← (S[lhs] + S[rhs])
9:             if sum < lookingLabel then
10:                lhs ← (lhs +1)
11:            end if
12:            if sum > lookingLabel then
13:                rhs ← (rhs −1)
14:            end if
15:            if sum = lookingLabel then
16:                c.saveTriplet(S[lhs], (t − S[rhs]), S[i])
17:                rhs ← (rhs −1)
18:                lhs ← (lhs +1)
19:            end if
20:        end while
21:    end for
22:    return c
23: end procedure
```

2. **Řešení úlohy 3-SUM pomocí metody hašování**
   Opět množinu S reprezentujeme pomocí datové struktury pole uloženou v proměnné

S. Dále vytvoříme prázdnou hašovací tabulku h. Vložíme všechny labely do hašovací tabulky. Pak pro každou dvojici labelů S[i] a S[j] začneme testovat, zda label s hodnotu $(t - S[i] - S[j])$ se nenachází v hašovací tabulce. Pokud takový label se nachází, zkontrolujeme, že tento label $(t - S[i] - S[j])$ je různý od labelů S[i] a S[j]. Pak labely $(t - S[i] - S[j])$, S[i] a S[j] zahrneme do nalezených kombinací. Uvádíme zde Algoritmus 14, jehož asymptotická složitost je $\mathcal{O}(n^2)$ kvůli tomu, že procházíme každou dvojici v poli S. Předpokládáme, že asymptotická složitost dotazu na existenci labelu $(t - S[i] - S[j])$ v hašovací tabulce je $\mathcal{O}(1)$. Prostorová složitost Algoritmu 14 je $\mathcal{O}(n)$. Pomocí stejného triku jako v popisu Algoritmu 10 můžeme snížit prostorovou složitost hašovací tabulky na $n + 1$ bitů s využitím bitového pole.

Algoritmus 14 Hledání řešení úlohy 3-SUM pomocí metody hašování

```
 1: procedure HASHING3-SUM([ ] S, t)
 2:     HashTable h
 3:     Combinations c
 4:     for i ← 1; i ≤ Lenght(S); i ← (i+1) do
 5:         h.Insert(S[i])
 6:     end for
 7:     for i ← 1; i ≤ Lenght(S)−2; i ← (i+1) do
 8:         for j ← i; j ≤ Lenght(S)−1; j ← (j+1) do
 9:             lookingNumber ← (t − S[i] −S[j])
10:             if h.contain(S[lookingNumber]) ∧ lookingNumber > S[j] then
11:                 c.saveTriplet(S[i], lookingNumber, S[j])
12:             end if
13:         end for
14:     end for
15:     return c
16: end procedure
```

5.4.4 4-SUM úloha

U úlohy 4-SUM už pouze ukážeme kód. Princip hledání všech čtyř prvkových množin S' se součtem t je analogický, jako u hledání všech k-prvkových podmnožin S' v úlohách 2-SUM a 3-SUM.

Poznámka 15 V proceduře *Labeling* Algoritmu 2, lze před zavolání procedury *ComputeCombination*, která začne řešit úlohu 4-SUM, určit, že aktuálně sestavený ohodnocený podgraf grafu v matici *solution* je ohodnocená komponenta souvislosti. Tato informace se netýká prvního zavolání procedury *ComputeCombination*, kdy je potřeba nalézt všechny možnosti, jak ohodnotit první label v matici *solution*, která je prázdná. Informací o tom, že se jedná o komponentu souvislosti, víme díky tomu, že současný label má minimální počet nepřiřazených sousedních labelů a byl vybrán pomocí procedury *NextLabel*, která hledá minimální nenulovou hodnotu v

poli *countOfMissingNeighbours*. Tento label má čtyři nepřiřazené sousední labely a menší nenulové hodnotě k 1, 2 a nebo 3 se v poli *countOfMissingNeighbours* aktuálně nenachází, jinak by je metoda *NextLabel* nalezla a vybrala pro další ohodnocení. Proto se musí jednat o komponentu souvislosti. Pokud tedy budeme chtít hledat pouze grafy obsahující komponenty souvislosti, je potřeba přidat omezující podmínky o počtu vyplněných labelů v matici *solution* před proceduru *ComputeCombination* v Algoritmu 2. Tím, že přidáme podmínky zkrátíme dobu výpočtů pro hledání určitého rozšířeného hendikepového ohodnocení. Pokud bychom hledali 4-pravidelné grafy obsahující pouze jednu komponentu souvislosti, je potřeba zakázat volání procedury *ComputeCombination* pro řešení úlohy 4-SUM kromě prvního volání.

1. **Řešení úlohy 4-SUM pomocí metody dvou ukazatelů**
   Asymptotická složitost Algoritmu 15 je $\mathcal{O}(n^3)$. Prostorová složitost Algoritmu 15 je $O(1)$.

Algoritmus 15 Hledání řešení úlohy 4-SUM pomocí metody dvou ukazatelů

```
1: procedure TwoPointers4-SUM([ ] S, t)
2:     Combinations c
3:     for i ← |S|; i ≥ 4;  i ← (i−1) do
4:         for j ←i−1; j ≥ 3;  j ← (j−1) do
5:             lhs ← 1
6:             rhs ←(j −1)
7:             lookingLabel← (t − S[i] - S[j])
8:             while lhs < rhs do
9:                 sum ← (S[lhs] + S[rhs])
10:                if sum < lookingLabel then
11:                    lhs ← (lhs +1)
12:                end if
13:                if sum > lookingLabel then
14:                    rhs ← (rhs −1)
15:                end if
16:                if sum = lookingLabel then
17:                    c.quadruplets(S[lhs], (t − S[rhs]), S[i], S[j])
18:                    rhs ← (rhs −1)
19:                    lhs ← (lhs +1)
20:                end if
21:            end while
22:        end for
23:    end for
24:    return c
25: end procedure
```

2. **Řešení úlohy 4-SUM pomocí metody hašování**
   Asymptotická složitost Algoritmu 16 je $\mathcal{O}(n^3)$. Prostorová složitost Algoritmu 16 je $O(n)$. Stejně i zde lze snížit prostorovou složitost Algoritmů 16 na $n+1$ bitů s využitím bitového pole.

Algoritmus 16 Hledání řešení úlohy 4-SUM pomocí metody hašování

```
1:  procedure HASHING4-SUM([ ] S, t)
2:      HashTable h
3:      Combinations c
4:      for i ← 1; i ≤ Lenght(S); i ← (i+1) do
5:          h.Insert(S[i])
6:      end for
7:      for i ← 1; i ≤ Lenght(S)−3; i ← (i+1) do
8:          for j ← i; j ≤ Lenght(S)−2; j ← (j+1) do
9:              for k ← j; k ≤ Lenght(S)−1; k ← (k+1) do
10:                 lookingNumber ← (t − S[i] −S[j])
11:                 if h.contain(S[lookingNumber]) ∧ lookingNumber > S[k] then
12:                     c.saveQuadruples(S[j], lookingNumber, S[i], S[k])
13:                 end if
14:             end for
15:         end for
16:     end for
17:     return c
18: end procedure
```

5.4.5 Procedura ComputeCombinations

Nakonec v této sekci uvedeme proceduru *ComputeCombinations* v Algoritmu 17, která se bude starat o zavolání příslušné metody řešení k-SUM. Pokud je potřeba hledat řešení k-SUM úlohy, kde $k = 1$, ověříme pouze, že proměnná *missNeighbours* je hodnota nějakého labelu a poté se zkontrolujeme, zda tento label má nastavenou hodnotu *true* v poli *canUseLabel*[*missNeighbours*]. Jedná se o 5. řádek Algoritmu 17. Pokud je potřeba řešit k-SUM úlohu, kde $k > 1$, potom je potřeba projít pole *canUseLabel* a uložit sestupně ty labely l do pole *arrayLabels*, které mají nastavenou hodnotu *true* v poli*canUseLabel*[l]. Jedná se o 10. až 16. řádek. Poznamenejme, že proměnnou i značíme index v poli *arrayLabels*. Na zbývajících řádcích se pak zavolá příslušná metoda řešení k-SUM úlohy pro konkretní k.

Algoritmus 17 Procedura ComputeCombinations

```
1: procedure ComputeCombination([ ] canUseLabel, missNeighbours, missingSum)
2:     Array arrayLabels[n]
3:     if missNeighbours= 1 then
4:         Combinations c
5:         if missNeighbours ∈ labels ∧ canUseLabel[missNeighbours] then
6:             c.save(missNeighbours)
7:             return c
8:         end if
9:         i ← 0
10:        for all label l ∈ labels do
11:            if canUseLabel[l] then
12:                i ← i + 1
13:                arrayLabels[i] ← l
14:            end if
15:        end for
16:        arrayLabels.setSize(i)
17:    end if
18:    if missNeighbours= 2 then
19:        return TwoPointers2-SUM(arrayLabels, missingSum)
20:    end if
21:    if missNeighbours= 3 then
22:        return TwoPointers3-SUM(arrayLabels, missingSum)
23:    end if
24:    if missNeighbours= 4 then
25:        return TwoPointers4-SUM(arrayLabels, missingSum)
26:    else
27:        Combinations c
28:        return c
29:    end if
30: end procedure
```

5.5 Paralelizace programu

Jedním z klíčových faktorů při hledání rozšířených hendikepových ohodnocení 4-pravidelných grafů je paralelizace. Jelikož hledání všech rozšířených hendikepových ohodnocení je hodně výpočetně náročná úloha a to zejména ve chvíli, kdy se prozkoumávají a testují možné kombinace sousedních labelů, paralelizace nabízí možnost, jak zkrátit výpočetní čas.

Nás napadla jedna přímočará varianta paralelního řešení, která si v prvním kroku předpočítá všechny možné kombinace sousedních labelů k prvnímu labelu v rozšířeném hendikepovém ohodnocení. Těmto předpočítaným kombinacím budeme říkat úkoly. Vytvoříme si frontu úkolu, do které vložíme všechny naše úkoly, což jsou možné kombinace možných sousedních labelů k prv-

nímu labelu. Pak podle vztahu

$$threadsCount = x_{CPU} - 1$$

vypočteme počet vláken, kde proměnná x_{CPU} je rovna počtu CPU jader dané počítačové architektury, na kterém náš program bude spuštěn. Tento počet budeme značit *threadsCount*. Důvodem, proč proměnná *threadsCount* není rovná x_{CPU} je ten, že chceme aby operační systém měl jedno jádro určené k řízení procesů a operačního systému. Dále je potřeba vytvořit *threadsCount* DataObjectLabeling objektů, které jsou potřebné při Algoritmu 1, protože budeme chtít, aby vlákna při paralelním běhu nezasahovaly do stejných datových struktur. Vyhneme se tím nechtěnému přepisování dat. Je potřeba ještě do implementovat proceduru *PrepareLabeling* Algoritmu 18, která se postará o vytvoření DataObjectLabeling objektu, do kterého vyplní čtveřici prvních labelů z úkolu do matice *solution* a poté spustí nad objektem DataObjectLabeling proceduru *Labeling*, která se postará o nalezení 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením pro zbylé labely. Cílem je, aby si vlákna brala úkoly z fronty úkolů, které se pak budou vykonávat souběžně. Tento model se nazývá Thread pool. Jedním z možných problémů, které mohou nastat při paralelním vykonávání, je tzv. problém race condition, českým ekvivalentním výrazem bychom tento termín přeložili jako souběh. Souběh je problém, ve kterém nastane chybný přístup ke sdíleným proměnným. V našem pojetí může souběh nastat, když zvyšujeme počet nalezených řešení a vypisujeme množinu sousedních labelů na terminál. Tomuto bloku kódu se říká kritická sekce. Proto je nutné tuto kritickou oblast opatřit mutexem. Mutex je objekt, který zamezí tomu, aby dvě a více vláken ve stejný čas přistoupilo ke stejným proměnným. V našem případě se jedná o aktualizaci počtu nalezených řešení a výpisu řešení, zamezíme tím nekonzistenci informací ve výpisu terminálu.

Důvodem proč jsme v sekci k-SUM uváděli Algoritmy 10 a 14, které pracují za pomocí metody hašování, je ten, že je možné také tyto algoritmy paralelizovat. V takovém případě je jedinou kritickou sekcí vkládání nové k-tice labelu do struktury *Combinations*.

5.5.1 Procedura PrepareLabeling

Cílem procedury PrepareLabeling je vytvořit objekt typu DataObjectLabelig, který v Algoritmu 18 značíme proměnou *d*. DataObjectLabeling v sobě uchovává proměnné, které jsem uvedli v Algoritmu 1, kromě proměnné *countOfUniqueSolutions*, která v programu zůstává jako globální proměnná. Už nebudeme uvádět explicitní definici každé procedury znovu, předáním parametru DataObjectLabeling do procedur se činnost těchto procedur nezmění. Změní se jenom to, že se upraví hodnoty ve struktuře DataObjectLabeling, nikoliv však globálně. Na 8. řádku doplníme sousední labely z kombinace *c* labelu *startingLabel* přes proceduru *MarkLabelsToSolution* z Algoritmu 3. Na 12. řádku najdeme prvek s nejmenší hodnotou nepřiřazených sousedů přes proceduru *SelectNextLabel* z Algoritmu 5. Pak na 13. řádku spustíme rekurzivní proceduru *La-*

beling a začne samotné hledání rozšířených hendikepových ohodnocení 4-pravidelného grafu na n vrcholech. Na 15. řádku odebereme kombinaci sousedních labelů, které jsme přiřadili labelu *startingLabel* pomocí procedury *UnmarkLabelsFromSolution* z Algoritmu 4.

Algoritmus 18 Procedura PreprareLabeling

```
1:  procedure PrepareLabeling(combination c, label startingLabel, mutex m )
2:      DataObjectLabeling d
3:      d.solution ← Matrix type of ℕ^{4×n}
4:      d.filledLabelsInSolution ← 0
5:      for all label l ∈ labels do
6:          d.missingNeighboursLabelsSum[l] ← (ℓ + l)
7:          d.countOfMissingNeighbours[l]← 4
8:      end for
9:      for all label neighbourLabels ∈ c do
10:         MarkLabelsToSolution(d, startingLabel, neighbourLabel)        ▷ Call procedure
11:     end for
12:     nextLabel ← SelectNextLabel(d)
13:     Labeling(nextLabel, d, m)                                         ▷ Call procedure
14:     for all label neighbourLabel ∈ combination do
15:         UnmarkLabelsFromSolution(d, startingLabel, neighbourLabel)    ▷ Call procedure
16:     end for
17: end procedure
```

5.5.2 Kritická sekce v proceduře *Labeling*

Jak již bylo zmíněno v úvodní sekci Paralelizace programu v proceduře *Labeling* z Algoritmu 2 se nachází kritická sekce, kterou je třeba vyřešit pomocí mutexů. Pokud dvě a více vláken ve stejný čas budou chtít aktualizovat počet nalezených rozšířených hendikepových ohodnocení a vypsat nové ohodnocení na terminál, mutex pustí jen jedno vlákno, které vykoná své změny a ve chvíli, kdy toto vlákno opustí tuto kritickou sekci, pustí další vlákno atd. Uvádíme pouze část kódu procedury *Labeling*, která se změní.

Algoritmus 19 Ošetření kritické sekce proceduře Labeling

```
1: procedure LABELING(currentLabel, DataObjectLabeling d, mutex m)
2:     ...
3:     if filledLabelsInSolution = 4n then
4:         if CheckSolution(d) then                          ▷ Call procedure
5:             m.Lock()
6:             countOfUniqueSolutions ← (countOfUniqueSolutions + 1)
7:             PrintSolution(d)                              ▷ Call procedure
8:             m.Unlock()
9:         end if
10:    else
11:        nextLabel ← SelectNextLabel(d)
12:        Labeling(nextLabel, d, m)                         ▷ Call recursion
13:    end if
14:    ...
15: end procedure
```

5.6 Dosažené výsledky

Pomocí programu implementujícího Algoritmus 8 byla zkoumána existence rozšířeného hendikepového ohodnocení 3-pravidelných a 4-pravidelných grafů na n vrcholech. Přehledný popis nalezených 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením je uveden v Tabulce 3. Pro nalezené 3-pravidelné grafy s rozšířeným hendikepovým ohodnocením uvádíme Tabulku 4. Symbolem n v Tabulkách 5, 4 a 3 označujeme počet vrcholů r-pravidelného grafu. Symbolem # označujeme počet nalezených grafů s rozšířeným hendikepovým ohodnocením. Tyto grafy našel program implementující Algoritmus 8. Pro grafy, které mají u počtu nalezených ohodnocení symbol x, je dokázáno, že takové r-pravidelné grafy nemohou existovat. Důvodem, proč v Tabulkách 3, 4 a 5 nejsou zobrazeny liché počty vrcholů je ten, že díky Větě 5 existuje rozšířené hendikepové ohodnocení pouze na sudém počtu vrcholů.

n	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
#	x	x	x	0	0	10	0	0	0	34	5748	120788	>1356	>1102	> 52458

Tabulka 3: Počet různých nalezených 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením za pomocí sestaveného programu

5.6.1 Nalezené 4-pravidelné grafy na 12 vrcholech s rozšířeným hendikepovým ohodnocením

Z Tabulky 3 vidíme, že nejmenší počet vrcholů 4-pravidelného grafu, pro který program implementující Algoritmus 8 našel rozšířené hendikepové ohodnocení, bylo až dvanáct vrcholů. Pro 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením na dvou a čtyřech vrcholech nemůže

existovat žádný 4-pravidelný graf s rozšířeným hendikepovým ohodnocením, protože neexistuje žádný 4-pravidelný graf na tomto počtu vrcholů. Nejmenší 4-pravidelný graf je až kompletní graf K_5 na 5 vrcholech. Na šesti vrcholech taktéž nemůže existovat žádný 4-pravidelný graf s rozšířeným hendikepovým ohodnocením díky Větě 12. Na osmi a deseti vrcholech program nenalezl žádný 4-pravidelný graf s rozšířeným hendikepovým ohodnocením. Na dvanácti vrcholech program našel deset různých ohodnocených grafů. Všechny tyto grafy byly isomorfní s grafem H. K ověření isomorfismu těchto nalezených grafů jsme využili matematický software Wolfram Mathematica 10. Pomocí těchto grafů se nám podařilo nalézt 8 různých nekonečných tříd 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením za podmínky, že počet vrcholů n je násobkem čísla dvanáct. Tyto nekonečné třídy byly podrobně popsány v kapitole 4.4.

5.6.2 Nalezené 4-pravidelné grafy na 20 vrcholech s rozšířeným hendikepovým ohodnocením

Druhým nejmenším počtem vrcholů z Tabulky 3, pro které program implementující Algoritmus 8 našel rozšířené hendikepové ohodnocení, bylo až dvacet vrcholů. To znamená, že pro 4-pravidelné grafy s počtem vrcholů 14, 16 a 18 program nenašel žádné 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením a tedy pokud je Algoritmus 8 korektní, tak grafy s rozšířeným hendikepovým ohodnocením na tomto počtu vrcholů neexistují. Mezi těmito třiceti čtyřmi ohodnocenými grafy byly nalezeny jen tři neisomorfní 4-pravidelné grafy.

5.6.3 Nalezené 4-pravidelné grafy na 22 vrcholech s rozšířeným hendikepovým ohodnocením

Třetím nejmenším počtem vrcholů 4-pravidelných grafů, na kterém program nalezl rozšířené hendikepové ohodnocení je dvacet dva. Zde program nalezl 5748 různých grafů s rozšířeným hendikepovým ohodnocením. Z těchto 5748 grafů je pouze 570 grafů tvořeno dvojicí komponent H a $H_{5,5}$ (5-crown graf Obrázek 6). Kde graf H má 12 vrcholů a 5-crown graf $H_{5,5}$ má 10 vrcholů. Oba tyto grafy jsou 4-pravidelné a v součtu dávají 22 vrcholů. K nalezení počtu komponent jsme sestavili program implementující algoritmus BFS (Breadth-first search) tzv. prohledávání do šířky. Ve zbylých 5178 grafů s rozšířeným hendikepovým ohodnocením se jedná o souvislé grafy. Myslíme si, že z těchto grafů, které jsou tvořeny dvěma komponentami H a $H_{5,5}$ půjde odvodit konstrukce těchto grafů i na větším počtu vrcholů. Můžeme vyslovit Hypotézu 1, že existuji 4-pravidelné grafy s rozšířeným hendikepovým ohodnocením, jestliže počet vrcholů $n \equiv 0 \pmod{22}$.

Hypotéza 1 *Rozšířené hendikepové ohodnocení 4-pravidelných grafů na n vrcholech existuje, jestliže platí*

$$n \equiv 0 \pmod{22}$$

.

5.6.4 Nalezené 4-pravidelné grafy na 24 vrcholech s rozšířeným hendikepovým ohodnocením

Čtvrtým nejnižším počtem vrcholů, na kterém program nalezl 120788 různých rozšířených hendikepových ohodnocení, bylo dvacet čtyři. Z toho počtu ohodnocených grafů je 3763 4-pravidelných grafů, které jsou tvořeny dvěma komponentami H, které nalezl program implementující Algoritmus 8. Ve zbylých 117025 různých případech se jednalo o souvislé grafy. Některé tyto 4-pravidelné graf, s rozšířeným hendikepovým ohodnocením umíme zkonstruovat díky důkazu Věty 14, kde n je násobkem čísla 12, což je číslo $n = 24$.

5.6.5 Nalezené 4-pravidelné grafy na 26, 28, 30 vrcholech s rozšířeným hendikepovým ohodnocením

Pro zbylé hodnoty n 4-pravidelných grafů 26, 28 a 30, které jsou uvedeny v Tabulce 3 jsme výpočet po několika hodinách přerušili a zapsali dosažené výsledky. Z pozorování v Tabulce 3, můžeme vyslovit Hypotézu 2, že rozšířené hendikepové ohodnocení existuje pro všechny 4-pravidelné grafy mající počet vrcholů sudý a větší nebo rovný dvaceti. Taktéž si můžeme všimnout z Tabulky 3, že se zvyšujícím se počtem vrcholů narůstá i počet různých 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením.

Hypotéza 2 *4-pravidelné grafy s rozšířeným hendikepovým ohodnocením existují, jestliže počet vrcholů* $n \geq 20$ *a zároveň* $n \equiv 0 \pmod 2$.

5.6.6 Nalezené rozšířené hendikepové ohodnocení na 3-pravidelných grafech

n	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
#	x	x	x	1	0	0	0	39	0	0	0	1763	0	0	586	433	10

Tabulka 4: Počet různých nalezených 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením za pomocí sestaveného programu

Dále zde uvedeme Tabulku 4 s počty nalezených 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením. Pro 3-pravidelné grafy na 2 vrcholech nemůže existovat žádný 3-pravidelný graf natož 3-pravidelný graf s rozšířeným hendikepovým ohodnocením. Pro počty vrcholů 4 a 6 existují Věty 9 a 10, které vyvrací existenci 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením. Pro grafy s $n = 10, 12, 14, 18, 20, 22, 26$ a 28 program nenalezl žádné 3-pravidelné grafy s rozšířeným hendikepovým ohodnocením. Pro 3-pravidelné grafy, kde počet vrcholů n je násobkem čísla 8, umíme zkonstruovat takový graf díky konstruktivnímu Důkazu Věty 11. Jednou zvláštností je, že pro třicet vrcholů 3-pravidelného grafu program nalezl 586 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením, kde všechny tyto grafy jsou tvořeny třemi komponentami Petersenovými grafy (viz Obrázek 5, kde je zobrazen Petersnův

graf). Můžeme vyslovit Hypotézu 3 o existenci 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením, jestliže počet vrcholů $n \equiv 0 \pmod{30}$.

Hypotéza 3 *3-pravidelný graf s rozšířeným hendikepovým ohodnocením existuje, jestliže počet vrcholů*

$$n \equiv 0 \pmod{30}.$$

Také i z Tabulky 4 můžeme vyslovit Hypotézu 4 o tom, že 3-pravidelné grafy s hendikepovým ohodnocením existují, jestliže počet vrcholů je sudý a větší nebo rovný 30.

Hypotéza 4 *Rozšířené hendikepové ohodnocení 3-pravidelných grafů na n vrcholech existuje, jestliže platí $n \geq 30$ a zároveň $n \equiv 0 \pmod{2}$.*

5.6.7 Nalezené 2-pravidelné grafy s rozšířeným hendikepovým ohodnocením

n	6	24	30	48	54	72
#	1	8	21	3040	20505	>2000000

Tabulka 5: Počet různých nalezených 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením za pomocí sestaveného programu

Uvádíme zde Tabulku 5 nalezených 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením. K nalezení těchto grafů jsme využili počítačový program, který implementoval modifikaci Algoritmu 8. Vidíme, že program našel rozšířené hendikepové ohodnocení a platí jedno z tvrzení Věty 8, že počet vrcholů 2-pravidelného grafu n musí splňovat $n \equiv 0, 6 \pmod{24}$. Nalezený počet 2-pravidelný grafů s rozšířeným hendikepovým ohodnocením byl zkontrolován s počtem 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením, který nalezl počítačový program sestavený mým vedoucím práce Doc. Petrem Kovářem, Ph.D. při zkoumání 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením, který využívá odlišného algoritmu k hledání těchto 2-pravidelných grafů s rozšířeným hendikepovým ohodnocením. Je zde silná evidence, že program implementující Algoritmus 8 pracuje správně, neboť počty se přesně shodují.

6 Závěr

Tato bakalářská práce se zabývá oblastí teorie grafů zvané ohodnocení grafů. Cílem zkoumání bylo rozšířené hendikepové ohodnocení, které zavedl Matěj Krbeček ve své diplomové práci [1]. Rozhodli jsme se navázat na tuto diplomovou práci Matěje Krbečka a pokusit se nalézt nové, dosud nepublikované výsledky, tykající se 3-pravidelných a 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením.

V úvodních kapitolách této bakalářské práce jsme zmínili teoretický rozbor potřebný k pochopení problematiky r-pravidelných grafů s rozšířeným hendikepovým ohodnocením a zmínili dosažené poznatky v této oblasti podle pokynů zadání této bakalářské práce.

Dalším z bodů bakalářské práce bylo uvést a naimplementovat algoritmus pro hledání tohoto typu ohodnocení. Tohoto cíle jsme dosáhli. Algoritmus 8 je detailně popsán v sekci 5 a naimplementován v jazyce C++11. Tento rekurzivní Algoritmus 8 slouží ke hledání všech 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na daném počtu vrcholů. Jedná se o polynomiální algoritmus, který v každém zanoření provede nejvýše $\mathcal{O}(n^3)$ operací pro hledání čtveřice vhodných labelů při sestavování 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením. Jednou z výhod Algoritmu 8 je možnost modifikovat pseudokód a využít tento algoritmus k hledání všech r-pravidelných grafů s rozšířeným hendikepovým nebo jen hendikepovým ohodnocením. Další z výhod Algoritmu 8 je možnost modifikovat kód a hledat pouze souvislé grafy nebo grafy tvořené komponentami souvislosti. Taktéž se nám podařilo naimplementovat paralelizaci Algoritmu 8, čímž jsme zrychlili hledání těchto r-pravidelných grafů s rozšířeným hendikepovým ohodnocením a splnili další z bodů v zadání této bakalářské práce. Implementovaná paralelizace je dobře škálovatelná, neboť vykonávání každé větve od kořene prohledávacího stromu pracuje téměř bez zbytečného čekání na další vlákno a v odděleném prostoru. Jediné čekání nastává při aktualizaci počtu nalezených 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením a jejich vypsání na výstup konzole. Proto, aby paralelizace byla co nejvíce efektivní, doporučujeme použít $(x_{CPU} - 1)$ vláken, kde x_{CPU} je počet CPU jader počítačové architektury. Změřené výsledky paralelizace jsou zobrazeny v příloze na Obrázcích 35 a 36.

Mezi dalšími body v bakalářské práci bylo ze získaných výsledků z počítačového programu, který implementoval Algoritmus 8, přijít s novou vlastní konstrukcí daného ohodnocení. Tento bod se nám podařilo splnit. S mým vedoucím práce doc. Petrem Kovářem, Ph.D. se nám podařilo společně nalézt 8 různých dosud nepublikovaných nekonečných tříd 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením za podmínky, že počet vrcholů n je násobkem čísla 12. Tento nový poznatek je obsažen ve Větě 14. Důkaz Věty 14 je detailně popsán v sekci 4.4.

Z dalších výsledků, které se nám podařilo společně nalézt, je taktéž nová, dosud nepubliko-

vaná třída 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením. Táto třída má počet vrcholů násobek čísla 8. Tento výsledek je shrnut do Věty 11, kde Důkaz Věty 11 je detailně popsán v sekci 4.3.

V této bakalářské práci jsou navíc uvedeny vlastní Věty 9 a 10 o neexistenci 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením na 4 a 6 vrcholech a Věty 12 a 13 o neexistenci 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na 6 a 8 vrcholech.

Díky této práci jsem se dozvěděl mnoho nových informací o grafovém ohodnocení r-pravidelných grafů. Taktéž mi tato práce poskytla příležitost podílet se na nových výzkumných výsledcích. Propojit získané výsledky s matematikou. Myslím si, že z velké části se mi podařilo splnit požadavky této bakalářské práce.

Mezi dalšími směry výzkumu, mohou být nové konstrukce r-pravidelných grafů s rozšířeným hendikepovým ohodnocením, či dokázání Hypotéz 2 nebo 4 o existenci 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením, kde počet vrcholů je sudý a větší nebo rovný dvaceti a nebo o existence 3-pravidelných grafů s rozšířeným hendikepovým ohodnocením, kde počet vrcholů je sudý a větší nebo rovný třiceti. Myslíme si, že uvedené 3-pravidelné grafy pro $n \leq 30$ existují a v sekci 5.6 jsme toto očekávání vyslovili jako hypotézu. Dalším směrem výzkumu může být vylepšení Algoritmu 8 a pokusit se o implementaci paralelního běhu na grafických kartách.

Literatura

[1] KRBEČEK, Matěj. *Zobecnění magických ohodnocení a jejich využití při losování sportovních turnajů* [online]. Ostrava, 2014 [cit. 2018-04-25]. Dostupné z: http://hdl.handle.net/10084/103803. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava.

[2] KOVÁŘ, Petr. *Teorie grafů, učební text*; [online]. 2018, [cit. 2018-04-05]. Dostupné z: http://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_rozsirene.pdf

[3] KOVÁŘ, Petr, Tereza KOVÁŘOVÁ, Bohumil KRAJC, Michal KRAVČENKO A Matěj KRBEČEK. *On regular handicap graphs*, manuscript.

[4] SHEPANIK, Aaron. *Graph Labelings and Tournament Scheduling.* MS Thesis, University of Minnesota 2015, http://hdl.handle.net/11299/174714.

[5] KOVÁŘOVÁ, Tereza, Petr KOVÁŘ a Bohumil KRAJC. *Handicap labelings of regular graphs*, [online]. Seminář diskrétní matematiky DiMaS 2012 [cit. 2018-04-07]. Dostupné z: http://graphs.vsb.cz/archive/seminar121022/grafy2012_TerezaKovar_DIMAS.pdf

[6] FRONČEK, Dalibor. *Handicap distance antimagic graphs and incomplete tournaments.* In: *AKCE International Journal of Graphs and Combinatorics.* vol. 10, iss. 2, s. 119-127, 2013 [cit. 2018-04-25] ISSN 0972-8600.

[7] *Crown graph. Wolfram mathworld* [online]. [cit. 2018-03-31]. Dostupné z: http://mathworld.wolfram.com/CrownGraph.html

[8] CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST a Clifford STEIN. *Introduction to Algorithms.* 3rd. Cambridge, MA, U.S.A: MIT Press, 2009. ISBN 978-0-262-53305-8.

[9] *3SUM.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2018-03-12]. Dostupné z: https://en.wikipedia.org/wiki/3SUM

[10] GAJENTAAN, Anka a Mark H. OVERMARS. *On a class of $O(n^2)$ problems in computational geometry* ☆. In: *Computational Geometry* [online]. 1995, s. 165-185 [cit. 2018-04-25]. ISSN 0925-7721. Dostupné z: http://www.sciencedirect.com/science/article/pii/0925772195000222

[11] Interview Question: *The Two Sum Problem.* [online], 2017 [cit. 2018-03-12]. Dostupné z: https://web.stanford.edu/class/cs9/sample_probs/TwoSum.pdf

[12] Two Sum Problem Analysis 2: *Not Unique and No Duplicates* [cit. 2018-03-12]. Dostupné z https://www.sigmainfy.com/blog/two-sum-problem-analysis-2-not-unique-no-duplicate.html

[13] ERICKSON, Jeff. *Lower Bounds for Linear Satisfiability Problems*. In: *Chicago Journal of Theoretical Computer Science* [online]. 1997, s. 388–395 [cit. 2018-04-25]. DOI: 10.1.1.47.2710. Dostupné z: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.2710&rep=rep1&type=pdf`

[14] DEMAINE, Erik. *6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs (Fall '14): 3SUM & APSP* [online]. 2014 [cit. 2018-03-16]. Dostupné z: `http://courses.csail.mit.edu/6.890/fall14/lectures/L21.html`

[15] GRØNLUND, Allan a Seth PETTIE. *Threesomes, Degenerates, and Love Triangles*. In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science* [online]. Philadelphia, PA, USA: IEEE Computer Society Press, 2014, s. 621-630 [cit. 2018-04-25]. DOI: 10.1109/FOCS.2014.72. ISBN 978-1-4799-6517-5. ISSN 0272-5428. Dostupné z: `https://arxiv.org/abs/1404.0799`

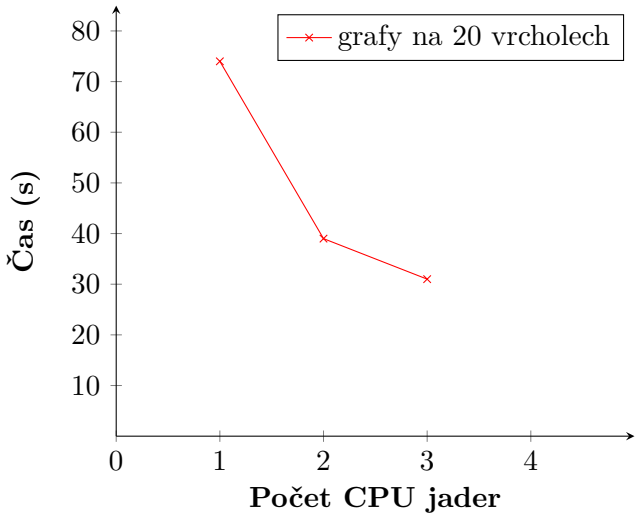
[16] GOLD, Omer a Micha SHARIR. *Improved Bounds for 3SUM, k-SUM, and Linear Degeneracy*. In: *25th Annual European Symposium on Algorithms (ESA 2017)* [online]. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, s. 1-13 [cit. 2018-04-25]. DOI: 10.4230/LIPIcs.ESA.2017.42. ISBN 978-3-95977-049-1. ISSN 1868-8969. Dostupné z: `http://drops.dagstuhl.de/opus/volltexte/2017/7836`

[17] MAREŠ, Martin a Tomáš VALLA; *Průvodce labyrintem algoritmů*, Edice CZ.NIC, Praha, 2017 [cit. 2018-04-08] Dostupné z: `https://knihy.nic.cz/files/edice/pruvodce_labyrintem_algoritmu.pdf`. ISBN 978-80-88168-19-5.

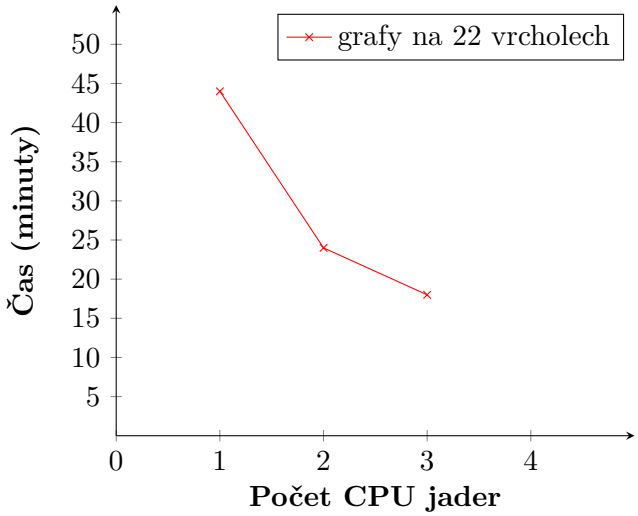
[18] WILLIAMS, Anthony. *C++ Concurrency IN ACTION Practical Multithreading*, MANNING, 2012. ISBN 978-19-33988-77-1.

A Paralelizace počítačového programu

K měření doby výpočtu programu, který implementoval Algoritmus 8 v jazyce C++11, jsme využili notebook MacBook Pro Late 13 s čtyřjádrovým CPU procesorem o taktu 2,4 GHz, Intel Core i5. Připomeňme, že měřené časy se na různých platformách mohou lišit. K měření doby výpočtu jsme použili hledání 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na 20 a 22 vrcholech. Výsledné grafy s naměřenými časy jsou zobrazeny na Obrázcích 35 a 36.



Obrázek 35: Měření doby výpočtu 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na 20 vrcholech



Obrázek 36: Měření doby výpočtu 4-pravidelných grafů s rozšířeným hendikepovým ohodnocením na 22 vrcholech